aws

Guía para desarrolladores

Amazon DynamoDB

Versión de API 2012-08-10

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon DynamoDB: Guía para desarrolladores

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Las marcas comerciales y la imagen comercial de Amazon no se pueden utilizar en relación con ningún producto o servicio que no sea de Amazon, de ninguna manera que pueda causar confusión entre los clientes y que menosprecie o desacredite a Amazon. Todas las demás marcas registradas que no son propiedad de Amazon son propiedad de sus respectivos propietarios, que pueden o no estar afiliados, conectados o patrocinados por Amazon.

Table of Contents

¿Qué es Amazon DynamoDB? 1
Características 2
Sin servidor 2
NoSQL 2
Totalmente administrado 2
Rendimiento de milisegundos de un solo dígito a cualquier escala 3
Casos de uso 3
Capacidades 4
Replicación multiactiva con tablas globales 4
Transacciones ACID 5
Captura de datos de cambios 5
Índices secundarios 5
Integraciones de servicios 5
Integraciones sin servidor 5
Importación y exportación de datos a Amazon S3 6
Integración sin ETL 6
Almacenamiento en caché 6
Seguridad 7
Resiliencia 7
Tablas globales 8
Copias de seguridad continuas y recuperación a un momento dado 8
Backup y restauración en diferido 8
Acceso a DynamoDB 9
Precios 9
Introducción 9
Introducción a DynamoDB 11
Recursos para usuarios nuevos 12
Prácticas recomendadas adicionales de Amazon DynamoDB para usuarios nuevos 13
Recursos de AWS CLI 13
Recursos de programación 14
Acceso a DynamoDB 14
Uso de la consola 15
Uso de AWS CLI 15
Uso de la API 18

Uso de NoSQL Workbench 18
Rangos de direcciones IP 20
Puntos de conexión de doble pila para el protocolo de Internet versión 6 (IPv6) 20
Requisitos previos 21
Configuración de DynamoDB 21
Configuración de DynamoDB (servicio web) 22
Configuración de la versión de DynamoDB local (versión descargable) 25
Paso 1: crear una tabla 51
Paso 2: escribir datos 105
Paso 3: leer datos 137
Paso 4: actualizar los datos 165
Paso 5: consultar los datos 199
Paso 6: (Opcional) Limpieza 237
Siguientes pasos 256
Console-to-Code 257
Funcionamiento 257
Beneficios del uso de Consola a código con DynamoDB 258
Ejemplos de casos de uso 258
Introducción 259
Funcionamiento 260
Hoja de referencia 260
Configuración inicial 260
SDK o CLI 261
Acciones básicas 261
Reglas de nomenclatura 262
Conceptos básicos sobre Service Quotas 263
Más información 265
Componentes básicos 265
Tablas, elementos y atributos 266
Clave principal 270
Índices secundarios 271
DynamoDB Streams 274
API de DynamoDB 276
Plano de control 276
Plano de datos 277
DynamoDB Streams 279

Transacciones ... 279
Tipos de datos y reglas de nomenclatura admitidos ... 280
Reglas de nomenclatura ... 280
Tipos de datos ... 281
Descriptores de tipos de datos ... 287
Clases de tablas de DynamoDB ... 287
Particiones y distribución de datos en DynamoDB ... 288
Distribución de datos: clave de partición ... 289
Distribución de datos: clave de partición y clave de clasificación ... 290
Información sobre cómo pasar de SQL a NoSQL ... 293
¿Base de datos relacional o NoSQL? ... 294
Acceso y autenticación ... 297
Creación de una tabla ... 300
Obtención de información sobre una tabla ... 302
Escritura de datos en una tabla ... 304
Lectura de datos de una tabla ... 309
Administrar índices ... 317
Modificación de los datos de una tabla ... 323
Eliminación de datos de una tabla ... 327
Eliminación de una tabla ... 329
Recursos y herramientas de aprendizaje de Amazon DynamoDB ... 330
Herramientas de programación y visualización ... 330
Orientación prescriptiva ... 331
Centro de conocimiento ... 332
Entradas de blog, repositorios y guías ... 333
Creación de modelos de datos y patrones de diseño ... 334
Cursos de formación técnica ... 334
Lecturas y escrituras ... 336
Coherencia de lectura de DynamoDB ... 336
Lecturas coherentes posteriores ... 336
Lecturas altamente coherentes ... 337
Coherencia de lectura de las tablas globales ... 337
Operaciones de lectura y escritura ... 337
Consumo de operaciones de lectura ... 338
Consumo de operaciones de escritura ... 340
Capacidad de rendimiento de DynamoDB ... 342

Modo bajo demanda ... 342
Modo aprovisionado ... 343
Modo de capacidad bajo demanda de DynamoDB ... 343
Unidades de solicitud de lectura y de escritura ... 345
Rendimiento inicial y propiedades de escalado ... 345
Rendimiento máximo de DynamoDB para las tablas bajo demanda ... 346
Modo de capacidad aprovisionada ... 348
Unidades de capacidad de lectura y escritura ... 350
Elección de los ajustes de rendimiento iniciales ... 350
Escalado automático de DynamoDB ... 351
Administración de la capacidad de rendimiento con escalado automático ... 352
Capacidad reservada ... 384
Rendimiento en caliente ... 385
Comprobación del rendimiento en caliente de la tabla ... 386
Aumento del rendimiento en caliente de la tabla ... 389
Creación de una tabla con rendimiento en caliente más alto ... 396
Escenarios de rendimiento en caliente ... 407
Capacidad de ampliación y de adaptación ... 409
Capacidad de ampliación ... 409
Capacidad de adaptación ... 410
Cambio de los modos de capacidad ... 411
Del modo aprovisionado al modo bajo demanda ... 412
Del modo bajo demanda al modo aprovisionado ... 414
Programación con DynamoDB ... 416
Información general sobre la compatibilidad de los SDK de AWS con DynamoDB ... 416
Compatibilidad del SDK con puntos de conexión basados en cuentas de AWS ... 418
Interfaces programáticas que funcionan con DynamoDB ... 419
Interfaces de programación de nivel superior ... 427
Ejecución de los ejemplos de código ... 490
API de bajo nivel ... 498
Programación con Python ... 504
Acerca de Boto ... 505
Documentación de Boto ... 505
Capas de clientes y recursos ... 506
Uso de batch_writer ... 509
Ejemplos de código adicionales ... 510

Seguridad de las sesiones y los subprocesos 510
Config 511
Gestión de errores 516
Registro 518
Enlaces de eventos 520
La paginación y el paginador 521
Esperadores 523
Programación con JavaScript 523
Acerca de AWS SDK para JavaScript 524
AWS SDK para JavaScript versión 3 524
Documentación de JavaScript 525
Capas de abstracción 525
Función de utilidad de serialización 528
Lectura de elementos 528
Escrituras condicionales 530
Paginación 531
Config 533
Esperadores 536
Gestión de errores 537
Registro 539
Consideraciones 540
Programación con AWS SDK for Java 2.x 541
Acerca del AWS SDK for Java 2.x 541
Introducción 542
Documentación del SDK para Java 2.x 552
Interfaces admitidas 552
Ejemplos de código adicionales 567
Programación síncrona y asíncrona 567
Clientes de HTTP 568
Config 570
Gestión de errores 577
ID de solicitud de AWS 578
Registro 579
Paginación 581
Anotaciones de clases de datos 583
Gestión de errores 583

Componentes de un error ... 583
Errores transaccionales ... 584
Mensajes y códigos de error ... 584
Control de errores en la aplicación ... 590
Reintentos de error y retroceso exponencial ... 591
Operaciones por lotes y control de errores ... 592
Cómo utilizar los AWS SDK ... 593
Uso de DynamoDB ... 595
Uso de tablas ... 595
Operaciones básicas en tablas ... 596
Aspectos que tener en cuenta a la hora de elegir una clase de tabla en DynamoDB ... 605
Etiquetas ... 606
Trabajo con tablas globales ... 612
Funcionamiento ... 613
Tutoriales: creación de tablas globales ... 623
Seguridad de las tablas globales ... 661
Tablas globales de facturación ... 674
Versiones de tablas globales ... 677
Prácticas recomendadas de tablas globales ... 689
Uso de elementos ... 690
Tamaños y formatos de elementos ... 692
Lectura de un elemento ... 693
Escritura de un elemento ... 694
Valores devueltos ... 696
Operaciones por lotes ... 698
Contadores atómicos ... 700
Escrituras condicionales ... 701
Uso de expresiones ... 708
Periodo de vida (TTL) ... 749
Consulta de tablas ... 780
Análisis de tablas ... 790
Lenguaje de consulta PartiQL ... 800
Uso de elementos: Java ... 848
Uso de elementos: .NET ... 880
Uso de índices ... 914
Índices secundarios globales ... 919

Índices secundarios locales ... 1021
Trabajo con transacciones ... 1077
Funcionamiento ... 1078
Uso de IAM con las transacciones ... 1087
Código de ejemplo ... 1091
Uso de secuencias ... 1095
Opciones ... 1096
Uso de Kinesis Data Streams ... 1098
Uso de DynamoDB Streams ... 1116
Aceleración en memoria con DAX ... 1201
Casos de uso de DAX ... 1202
Notas de uso de DAX ... 1203
Funcionamiento ... 1204
Cómo procesa DAX las solicitudes ... 1206
Caché de elementos ... 1208
Caché de consultas ... 1209
Componentes del clúster de DAX ... 1210
Nodos ... 1210
Clústeres ... 1211
Regiones y zonas de disponibilidad ... 1212
Grupos de parámetros ... 1213
Grupos de seguridad ... 1213
ARN del clúster ... 1213
Punto de conexión de clúster ... 1214
Puntos de conexión del nodo ... 1214
Grupos de subredes ... 1214
Eventos ... 1215
Periodo de mantenimiento ... 1215
Creación de un clúster de DAX ... 1217
Creación de un rol de servicio de IAM para que DAX obtenga acceso a DynamoDB ... 1217
DAX e IPv6 ... 1219
Uso de AWS CLI ... 1221
Uso de la consola ... 1228
Modelos de coherencia ... 1233
Coherencia entre los nodos de los clústeres de DAX ... 1234
Comportamiento de la caché de elementos de DAX ... 1234

Comportamiento de la caché de consulta de DAX ... 1238
Lecturas altamente coherentes y transaccionales ... 1239
Almacenamiento en caché negativo ... 1239
Estrategias de escritura ... 1240
Desarrollo con el cliente de DAX ... 1243
Tutorial: Ejecución de un ejemplo de aplicación ... 1244
Modificación de una aplicación existente para que use DAX ... 1310
Administración de los clústeres de DAX ... 1311
Permisos de IAM para administrar un clúster de DAX ... 1311
Escalado de un clúster de DAX ... 1314
Personalización de las configuraciones de un clúster de DAX ... 1316
Configuración de los ajustes de TTL ... 1317
Compatibilidad con el etiquetado en DAX ... 1319
AWS CloudTrailIntegración de ... 1320
Eliminación de un clúster de DAX ... 1320
Supervisión de DynamoDB Accelerator ... 1320
Herramientas de supervisión de DAX ... 1321
Monitoreo con CloudWatch ... 1323
Registro de operaciones de DAX con AWS CloudTrail ... 1351
Instancias ampliables DAX T3/T2 ... 1351
Familia de instancias DAX T2 ... 1351
Familia de instancias DAX T3 ... 1352
Control de acceso a DAX ... 1353
Rol de servicio de IAM para DAX ... 1354
Política de IAM que permite el acceso a un clúster de DAX ... 1356
Caso práctico: acceso a DynamoDB y DAX ... 1357
Acceso a DynamoDB, pero no con DAX ... 1358
Acceder a DynamoDB y DAX ... 1360
Acceso a DynamoDB a través de DAX, pero sin acceso directo a DynamoDB ... 1366
Cifrado en reposo de DAX ... 1369
Habilitación del cifrado en reposo mediante la Consola de administración de AWS ... 1371
Cifrado en tránsito de DAX ... 1372
Uso de roles vinculados a servicios para DAX ... 1373
Permisos de roles vinculados a servicios para DAX ... 1374
Creación de un rol vinculado a un servicio para DAX ... 1375
Edición de un rol vinculado a un servicio para DAX ... 1375

Eliminación de un rol vinculado a un servicio para DAX ... 1376
Acceso a DAX a través de las cuentas de AWS ... 1377
Configurar IAM ... 1378
Configurar una VPC ... 1381
Modificar el cliente de DAX para permitir el acceso entre cuentas ... 1383
Guía de tamaño del clúster de DAX ... 1388
Información general ... 1388
Estimación del tráfico ... 1389
Prueba de carga ... 1390
Modelado de datos ... 1392
Uso de colecciones de elementos ... 1393
Acelerar las consultas mediante la organización de los datos con colecciones de elementos ... 1395
Principios básicos del modelado de datos ... 1395
Diseño de tabla única ... 1396
Diseño de tabla múltiple ... 1399
Componentes del modelado de datos ... 1400
Clave de clasificación compuesta ... 1401
Tenencia múltiple ... 1403
Índices dispersos ... 1404
Tiempo de vida ... 1405
Archivado de tiempo de vida ... 1407
Particionamiento vertical ... 1407
Partición de escritura ... 1410
Paquetes de diseño de esquemas de modelado de datos ... 1411
Requisitos previos ... 1412
Red social ... 1413
Perfil de juego ... 1423
Sistema de administración de reclamaciones ... 1432
Pagos periódicos ... 1451
Actualizaciones de estado de los dispositivos ... 1456
Tienda en línea ... 1470
Modelos relacionales ... 1494
Modelos tradicionales de bases de datos relacionales ... 1495
Cómo DynamoDB elimina la necesidad de las operaciones JOIN ... 1497
Cómo las transacciones de DynamoDB eliminan la sobrecarga del proceso de escritura ... 1498

Primeros pasos ... 1499
Ejemplo ... 1501
Migración a DynamoDB ... 1507
Razones para migrar ... 1507
Consideraciones a la hora de migrar ... 1509
Funcionamiento ... 1511
Herramientas de migración ... 1512
Elección de una estrategia de migración ... 1512
Migración sin conexión ... 1515
Migración híbrida ... 1517
En línea: migración de cada tabla de forma individual ... 1519
En línea: migración con una tabla provisional personalizada ... 1520
NoSQL Workbench ... 1523
Descargar ... 1524
Instalación ... 1526
Modelador de datos ... 1533
Creación de un nuevo modelo ... 1533
Importación de un modelo existente ... 1540
Exportación de un modelo ... 1543
Edición de un modelo existente ... 1545
Visualizador de datos ... 1549
Incorporación de datos de muestreo ... 1549
Importación desde CSV ... 1552
Facetas ... 1553
Vista agregada ... 1556
Confirmación de un modelo de datos ... 1557
Generador de operaciones ... 1560
Conexión a conjuntos de datos ... 1561
Generación de operaciones ... 1562
Clonación de tablas ... 1574
Exportación a CSV ... 1576
Modelos de datos de ejemplo ... 1577
Modelo de datos de empleados ... 1577
Modelo de datos del foro de debate ... 1578
Modelo de datos de biblioteca de música ... 1578
Modelo de datos de la estación de esquí ... 1579

Modelo de datos de ofertas de tarjetas de crédito 1579
Modelo de datos de marcadores 1580
Historial de versiones 1580
Copia de seguridad y restauración 1588
Copias de seguridad a un momento dado 1590
Antes de empezar 1590
Habilitación de la recuperación en un momento dado 1591
Copias de seguridad bajo demanda 1596
Funcionamiento 1596
Copia de seguridad de una tabla 1600
Restauración de una tabla 1602
Eliminación de una copia de seguridad de tabla 1608
Uso de IAM 1609
Facturación de copias de seguridad 1617
Funcionamiento 1617
Ejemplo de facturación de copias de seguridad de DynamoDB 1618
Restauraciones 1623
Restauración de una tabla mediante la recuperación a un momento dado 1623
Restauración de una tabla de DynamoDB a un momento específico 1625
Uso de AWS Backup 1631
Funcionamiento 1633
Creación de copias de seguridad 1636
Copiar copias de seguridad 1637
Restauración de una tabla 1639
Eliminación de copias de seguridad 1640
Copias de seguridad bajo demanda administradas por AWS Backup frente a DynamoDB . 1640
Ejemplos de código 1642
Conceptos básicos 1655
Introducción a DynamoDB 1656
Conceptos básicos 1666
Acciones 1823
Escenarios 2236
Aceleración de lecturas con DAX 2238
Escenarios avanzados de índice secundario global 2247
Creación de una aplicación para enviar datos a una tabla de DynamoDB 2250
Comparación de varios valores con un solo atributo 2251

Actualización condicional del TTL de un elemento 2274
Conexión a una instancia local 2281
Recuento de operadores de expresión 2282
Creación de una API REST para realizar un seguimiento de datos de COVID-19 2315
Creación de una aplicación de mensajería 2316
Creación de una aplicación sin servidor para administrar fotos 2317
Creación de una tabla con índice secundario global 2321
Creación de una tabla con rendimiento en caliente habilitado 2325
Creación de una aplicación web para hacer un seguimiento de los datos de DynamoDB ... 2335
Creación una aplicación de chat de websocket 2337
Creación de un elemento con un TTL 2338
Creación y administración de tablas globales de MRSC 2345
Creación y administración de tablas globales que muestren MREC 2365
Eliminación de datos mediante DELETE de PartiQL 2375
Detección de EPI en imágenes 2381
Inserción de datos mediante INSERT de PartiQL 2382
Invocación de una función de Lambda desde un navegador 2386
Administración de índices secundarios globales 2387
Administración de las políticas basadas en recursos 2389
Supervisión del rendimiento de DynamoDB 2392
Realización de operaciones de consulta avanzadas 2393
Realización de operaciones de lista 2428
Realización de operaciones de asignación 2461
Realización de operaciones de conjunto 2485
Consultar una tabla mediante lotes de instrucciones PartiQL 2515
Consultar una tabla con PartiQL 2578
Consulta de una tabla mediante un índice secundario global 2632
Consulta de una tabla mediante una condición begins_with 2641
Consulta de una tabla mediante un intervalo de fechas 2647
Consulta de una tabla con una expresión de filtro compleja 2656
Consulta de una tabla con una expresión de filtro dinámica 2664
Consulta de una tabla con una expresión de filtro y limite 2673
Consulta de una tabla con atributos anidados 2678
Consulta de una tabla con paginación 2687
Consulta de una tabla con lecturas altamente coherentes 2698
Consulta de datos mediante SELECT de PartiQL 2703

Consulta de elementos de TTL ... 2708
Consulta de tablas mediante patrones de fecha y hora ... 2713
Guarde EXIF y otra información de la imagen ... 2727
Configuración del control de acceso basado en atributos ... 2728
Descripción del orden de las expresiones de actualización ... 2731
Actualización de la configuración de rendimiento en caliente de una tabla ... 2762
Actualización del TTL de un elemento ... 2768
Actualización de datos mediante UPDATE de PartiQL ... 2772
Uso de API Gateway para invocar una función de Lambda ... 2779
Usar Step Functions para invocar funciones de Lambda ... 2781
Utilizar un modelo de documento ... 2782
Utilizar un modelo de persistencia de objetos de alto nivel ... 2798
Uso de operaciones de contador atómico ... 2807
Uso de operaciones condicionales ... 2830
Uso de nombres de atributo de expresión ... 2860
Uso de eventos programados para invocar una función de Lambda ... 2888
Trabajo con índices secundarios locales ... 2890
Trabajo con Streams y tiempo de vida ... 2893
Trabajo con tablas globales y coherencia final de replicación de varias regiones (MREC) ... 2897
Trabajo con el etiquetado de recursos ... 2900
Trabajo con el cifrado de tablas ... 2903
Ejemplos de tecnología sin servidor ... 2905
Invocación de una función de Lambda desde un desencadenador de DynamoDB ... 2905
Notificación de los errores de los elementos del lote de las funciones de Lambda con un desencadenador de DynamoDB ... 2915
Contribuciones de la comunidad de AWS ... 2926
Cómo crear y probar una aplicación sin servidor ... 2926
Seguridad ... 2929
AWSPolíticas administradas por ... 2930
Políticas gestionadas de AWS ... 2930
Política administrada de AWS: AmazonDynamoDBFullAccess_v2 ... 2931
AmazonDynamoDBReadOnlyAccess ... 2931
Actualizaciones de DynamoDB en las políticas administradas por AWS ... 2933
Políticas basadas en recursos ... 2935
Crear tabla ... 2936
Asociación de una política basada en recursos ... 2942

Asociación de una política a una secuencia ... 2947
Eliminación de una política basada en recursos ... 2950
Acceso entre cuentas ... 2950
Bloquear acceso público ... 2952
Operaciones de la API ... 2955
Autorización de IAM ... 2963
Ejemplos ... 2964
Consideraciones ... 2972
Prácticas recomendadas ... 2973
Control de acceso basado en atributos ... 2974
¿Por qué debo utilizar ABAC? ... 2976
Claves de condición ... 2977
Consideraciones ... 2978
Habilitación de ABAC en DynamoDB ... 2978
Uso de ABAC ... 2981
Ejemplos de casos de uso ... 2983
Solución de problemas ... 2996
Protección de datos ... 2998
Cifrado en reposo ... 2999
Protección de las conexiones de DynamoDB ... 3026
IAM ... 3028
Identity and Access Management ... 3029
Uso de condiciones ... 3060
Validación de conformidad ... 3087
Resiliencia ... 3087
Seguridad de la infraestructura ... 3088
Uso de puntos de conexión de VPC ... 3089
AWS PrivateLink para DynamoDB ... 3099
Tipos de puntos de conexión de Amazon VPC ... 3100
Consideraciones sobre el uso de AWS PrivateLink para Amazon DynamoDB ... 3101
Creación de un punto de conexión de VPC de Amazon ... 3101
Acceso a los puntos de conexión de la interfaz de Amazon DynamoDB ... 3102
Acceso a tablas de DynamoDB y operaciones de la API de control desde los puntos de conexión de la interfaz de DynamoDB ... 3102
Actualización de una configuración DNS en las instalaciones ... 3104
Creación de una política de punto de conexión de Amazon VPC ... 3107

Uso de puntos de conexión de DynamoDB con acceso privado de Consola de administración de AWS ... 3108
AWS PrivateLink para DynamoDB Streams ... 3108
AWS PrivateLink para DAX ... 3115
Configuración y análisis de vulnerabilidades ... 3118
Prácticas recomendadas de seguridad ... 3118
Prácticas recomendadas de seguridad preventivas ... 3119
Prácticas recomendadas de detección de seguridad ... 3121
Supervisión y registro ... 3126
Plan de monitoreo ... 3126
Referencia de rendimiento ... 3126
Servicios integrados ... 3127
Herramientas de supervisión automatizadas ... 3127
Supervisión de métricas ... 3128
¿Cómo uso las métricas de DynamoDB? ... 3128
Visualizar métricas en la consola de CloudWatch ... 3130
Visualización de las métricas en la AWS CLI ... 3130
Métricas y dimensiones ... 3131
Creación de alarmas de CloudWatch en DynamoDB ... 3162
Operaciones de registro ... 3166
Información de DynamoDB en CloudTrail ... 3167
Descripción de las entradas del archivo de registros de DynamoDB ... 3171
Contributor Insights ... 3190
Funcionamiento ... 3190
Introducción ... 3201
Uso de IAM ... 3211
Prácticas recomendadas ... 3218
Diseño NoSQL ... 3219
NoSQL versus RDBMS ... 3219
Dos conceptos clave ... 3220
Enfoque general ... 3220
NoSQL Workbench ... 3221
El enfoque Well-Architected de DynamoDB ... 3222
Optimización de costos ... 3222
Realización de la revisión del enfoque Well-Architected de Amazon DynamoDB ... 3274
Los pilares del enfoque Well-Architected de Amazon DynamoDB ... 3274

Diseño de claves de partición ... 3277
Distribución de cargas de trabajo ... 3278
Partición de escritura ... 3280
Carga de datos de forma eficiente ... 3281
Diseño de la clave de clasificación ... 3283
Control de versiones ... 3284
Índices secundarios ... 3285
Directrices generales ... 3286
Índices dispersos ... 3289
Agregación ... 3291
Sobrecarga de GSI ... 3292
Partición de ISG ... 3294
Creación de una réplica ... 3299
Elementos grandes ... 3300
Compresión ... 3301
Particionamiento vertical ... 3301
Uso de Amazon S3 ... 3302
Datos de serie temporal ... 3302
Patrón de diseño de los datos de serie temporal ... 3303
Ejemplos de tablas de serie temporal ... 3303
Relaciones de varios a varios ... 3304
Listas de adyacencia ... 3305
Gráficos materializados ... 3306
Consulta y análisis ... 3311
Rendimiento del análisis ... 3311
Evitar los picos ... 3312
Exámenes en paralelo ... 3315
Diseño de tabla ... 3316
Uso de tablas globales ... 3316
Datos clave ... 3317
Datos clave acerca de la MREC ... 3318
Datos clave acerca de la MRSC ... 3319
Casos de uso ... 3322
Modos de escritura ... 3323
Estrategias de enrutamiento de DynamoDB ... 3329
Procesos de evacuación ... 3335

Planificación de la capacidad de rendimiento ... 3339
Lista de verificación de preparación ... 3341
Conclusión y recursos ... 3350
Plano de control ... 3351
Operaciones de datos masivos ... 3351
Actualización por lotes condicional ... 3352
Operaciones masivas eficientes ... 3357
Implementación del control de versiones ... 3359
Cuándo utilizar este patrón ... 3359
Diseño de patrón ... 3360
Uso del patrón ... 3361
Informes de facturación y de uso ... 3363
Capacidad de desempeño ... 3367
Transmisión ... 3371
Almacenamiento ... 3372
Copia de seguridad y restauración ... 3373
Transferencia de datos ... 3377
CloudWatch ... 3378
DAX ... 3379
Migración de una tabla de DynamoDB de una cuenta a otra ... 3380
Migración de una tabla mediante AWS Backup para la copia de seguridad y la restauración entre cuentas ... 3381
Migración de una tabla mediante la exportación a S3 y la importación desde S3 ... 3384
Recomendaciones de DAX ... 3386
Evaluación de la idoneidad de DAX ... 3387
Configuración del cliente de DAX ... 3390
Configuración del clúster de DAX ... 3391
Dimensionamiento del clúster de DAX ... 3398
Implementación de un clúster ... 3405
Operaciones de clúster ... 3407
Monitoreo de DAX ... 3410
Uso de DynamoDB con otros servicios de AWS ... 3413
Integración con Amazon Cognito ... 3413
Integración con Amazon Redshift ... 3416
Consideraciones sobre la integración entre regiones con CMK ... 3416
Integración sin ETL con Amazon Redshift ... 3420

Carga de datos desde DynamoDB en Amazon Redshift con COPY 3429
Integración con Amazon EMR 3430
Información general 3431
Tutorial: Uso de Amazon DynamoDB y Apache Hive 3432
Creación de una tabla externa en Hive 3441
Procesamiento de instrucciones de HiveQL 3445
Consulta de datos en DynamoDB 3446
Copia de datos en y desde Amazon DynamoDB 3448
Ajuste del rendimiento 3463
Integración con S3 3469
Importación desde Amazon S3 3469
Exportar a Amazon S3. 3490
Integración con Amazon SageMaker Lakehouse 3513
Integración sin ETL con Amazon SageMaker Lakehouse 3513
Integración con Amazon OpenSearch Service 3515
Funcionamiento 3516
Creación de una integración 3517
Siguientes pasos 3517
Gestión de cambios importantes 3517
Integración sin ETL con OpenSearch Service 3521
Integración con Amazon EventBridge 3524
Funcionamiento 3525
Creación de una integración a través de la consola 3526
Siguientes pasos 3529
Integración con Amazon MSK 3529
Funcionamiento 3530
Integración de ejemplo 3531
Pasos a seguir a continuación 3537
Prácticas recomendadas de integración 3538
Creación de una instantánea 3538
Captura de datos de cambios 3538
Uso de la IA generativa con DynamoDB 3540
Casos de uso de IA generativa para DynamoDB 3540
Blogs de IA generativa para DynamoDB 3541
Uso de la integración sin ETL de DynamoDB con el servicio OpenSearch 3541
Cuotas y restricciones 3543

Realización de tareas de administración de cuotas 3543
Acceso a las cuotas de DynamoDB 3543
Visualización de las cuotas actuales en la consola 3544
Visualización de las cuotas actuales mediante AWS CLI 3544
Solicitud de aumento de cuota 3546
Cuotas 3547
Rendimiento de lectura o escritura 3548
Capacidad reservada 384
Tablas 3551
Tablas globales 3551
Índices secundarios 3553
Atributos de índice secundario proyectados 3553
DynamoDB Streams 3553
Importación desde Amazon S3 3554
Exportar tablas a Amazon S3 3554
Copia de seguridad y restauración 3554
Contributor Insights 3554
Restricciones 3554
Modo de capacidad de lectura/escritura 3555
Índices secundarios 3553
Claves de partición y claves de clasificación 3556
Reglas de nomenclatura 3557
Tipos de datos 3558
Items 3559
Atributos 3560
Parámetros de expresión 3560
Transacciones de DynamoDB 3561
DynamoDB Streams 3562
DynamoDB Accelerator (DAX) 3562
Restricciones específicas de la API 3563
Cifrado en reposo en DynamoDB 3565
Referencia de la API 3567
Solución de problemas 3568
Errores del servidor internos 3568
Investigación de los errores del servidor internos 3569
Minimización del impacto de los errores del servidor internos 3570

Mejora del conocimiento operativo ... 3570
Latencia ... 3574
Limitación ... 3578
Diagnóstico de limitación ... 3580
Guía de resolución ... 3587
Limitación de escritura de GSI y contrapresión ... 3614
Métricas de limitación de CloudWatch ... 3616
Apéndice ... 3619
Solución de problemas de establecimiento de conexiones SSL/TLS con DynamoDB ... 3619
Probar su aplicación o servicio ... 3619
Probar el navegador del cliente ... 3620
Actualización del cliente de aplicación de software ... 3620
Actualización del navegador del cliente ... 3621
Actualización manual su paquete de certificados ... 3621
Tablas y datos de ejemplo para usar en DynamoDB ... 3622
Ejemplos de archivos de datos ... 3623
Creación de ejemplos de tablas y carga de datos ... 3636
Creación de ejemplos de tablas y carga de datos: Java ... 3636
Creación de ejemplos de tablas y carga de datos: .NET ... 3646
Aplicación de ejemplo con AWS SDK for Python (Boto3) ... 3658
Paso 1: implementar y probar localmente ... 3660
Paso 2: examinar el modelo de datos y los detalles de implementación ... 3664
Paso 3: implementar en producción ... 3675
Paso 4: limpie los recursos ... 3685
Palabras reservadas en DynamoDB ... 3685
AWSEjemplos de SDK de para Java 1.x ... 3698
DAX y Java SDK v1 ... 3699
Modificación de una aplicación SDK para Java 1.x existente para que use DAX ... 3711
Consulta de índices secundarios globales con SDK para Java 1.x ... 3716
Ejemplos de AWS SDK para Go 1.x ... 3720
Go y DAX ... 3720
Ejemplos de AWS SDK para Node.js 2.x ... 3723
Node.js y DAX ... 3723
Historial de revisión ... 3734
Actualizaciones anteriores ... 3775
Características heredadas ... 3812

Versión 2017.11.29 (heredada) de las tablas globales 3812
Funcionamiento 3813
Prácticas recomendadas y requisitos 3818
Creación de una tabla global 3823
Monitoreo de tablas globales 3827
Uso de IAM con tablas globales 3829
Versión anterior de la API de DynamoDB de bajo nivel (2011-12-05) 3832
BatchGetItem 3833
BatchWriteItem 3841
CreateTable 3849
DeleteItem 3857
DeleteTable 3864
DescribeTables 3868
GetItem 3872
ListTables 3876
PutItem 3879
Consultar 3887
Examen 3905
UpdateItem 3926
UpdateTable 3936
Parámetros condicionales de DynamoDB heredados 3941
AttributesToGet 3943
AttributeUpdates 3944
ConditionalOperator 3947
Expected 3948
KeyConditions 3954
QueryFilter 3957
ScanFilter 3959
Escritura de condiciones con parámetros heredados 3961

¿Qué es Amazon DynamoDB?

Amazon DynamoDB es una base de datos NoSQL distribuida sin servidor completamente administrada con un rendimiento de milisegundos de un solo dígito a cualquier escala.

DynamoDB satisface sus necesidades para superar las complejidades operativas y de escalado de las bases de datos relacionales. DynamoDB está diseñado específicamente y optimizado para las cargas de trabajo operativas que requieren un rendimiento uniforme a cualquier escala. Por ejemplo, DynamoDB ofrece un rendimiento uniforme en milisegundos de un solo dígito en un caso de uso de un carrito de la compra, independientemente de si tiene 10 o 100 millones de usuarios. Lanzado en 2012, DynamoDB sigue ayudándole a dejar de usar bases de datos relacionales al tiempo que reduce los costos y mejora el rendimiento a escala.

Clientes de todos los tamaños, sectores y regiones geográficas utilizan DynamoDB para crear aplicaciones modernas sin servidor que puedan empezar con pocos recursos y escalarse en todo el mundo. DynamoDB se escala para admitir tablas de prácticamente cualquier tamaño y, al mismo tiempo, proporciona un rendimiento uniforme de milisegundos de un solo dígito y alta disponibilidad.

Para eventos como el Amazon Prime Day, DynamoDB potencia numerosas instalaciones y sistemas de Amazon con mucho tráfico, como Alexa, los sitios de Amazon.com y todos los centros logísticos de Amazon. Para este tipo de eventos, las API de DynamoDB han gestionado billones de llamadas desde las instalaciones y los sistemas de Amazon. DynamoDB atiende continuamente a cientos de clientes con tablas que tienen un tráfico máximo de más de medio millón de solicitudes por segundo. También atiende a cientos de clientes cuyos tamaños de tabla superan los 200 TB y procesa más de mil millones de solicitudes por hora.

Temas

- Características de DynamoDB
- Casos de uso de DynamoDB
- Capacidades de DynamoDB
- Integraciones de servicios
- Seguridad
- Resiliencia
- Acceso a DynamoDB
- Precios de DynamoDB

- Introducción a DynamoDB

Características de DynamoDB

Sin servidor

Con DynamoDB, no necesita aprovisionar ningún servidor ni aplicar parches, administrar, instalar, mantener ni operar ningún software. DynamoDB proporciona mantenimiento sin ningún tiempo de inactividad. No tiene versiones (principal, secundaria o de parche) y no hay periodos de mantenimiento.

El modo de capacidad bajo demanda de DynamoDB ofrece precios de pago por uso para las solicitudes de lectura y escritura, por lo que solo pagará por lo que utilice. Con la característica bajo demanda, DynamoDB escala o reduce verticalmente al instante las tablas para ajustarse a la capacidad y mantiene el rendimiento sin necesidad de administración. También se reduce verticalmente a cero, por lo que no tendrá que pagar por el rendimiento si su tabla no tiene tráfico y no hay arranques en frío.

NoSQL

Como base de datos NoSQL, DynamoDB está diseñada específicamente para ofrecer un mejor rendimiento, escalabilidad, capacidad de administración y flexibilidad en comparación con las bases de datos relacionales tradicionales. Para admitir una amplia variedad de casos de uso, DynamoDB permite modelos de datos de documentos y de valores clave.

A diferencia de las bases de datos relacionales, DynamoDB no admite el operador JOIN. Se recomienda desnormalizar el modelo de datos para reducir los viajes de ida y vuelta a la base de datos y la potencia de procesamiento necesaria para responder a las consultas. Como base de datos NoSQL, DynamoDB proporciona una sólida coherencia de lectura y transacciones ACID para crear aplicaciones empresariales.

Totalmente administrado

Al ser un servicio de bases de datos totalmente administrado, DynamoDB se ocupa del trabajo pesado e indiferenciado que supone administrar una base de datos para que usted pueda centrarse en generar valor para sus clientes. Se encarga de la instalación, las configuraciones, el mantenimiento, la alta disponibilidad, el aprovisionamiento de hardware, la seguridad, las copias de seguridad, la monitorización y mucho más. Esto garantiza que, al crear una tabla de DynamoDB,

esté lista al instante para las cargas de trabajo de producción. DynamoDB mejora constantemente su disponibilidad, fiabilidad, rendimiento, seguridad y funcionalidad sin necesidad de actualizaciones ni tiempo de inactividad.

Rendimiento de milisegundos de un solo dígito a cualquier escala

DynamoDB se creó específicamente para mejorar el rendimiento y la escalabilidad de las bases de datos relacionales a fin de ofrecer un rendimiento de milisegundos de un solo dígito a cualquier escala. Para lograr esta escala y rendimiento, DynamoDB está optimizado para cargas de trabajo de alto rendimiento y proporciona API que fomentan el uso eficiente de las bases de datos. Omite las características que son ineficientes y que no ofrecen rendimiento a gran escala como, por ejemplo, las operaciones JOIN. DynamoDB ofrece un rendimiento constante de milisegundos de un solo dígito para su aplicación, tanto si tiene 100 como 100 millones de usuarios.

Casos de uso de DynamoDB

Clientes de todos los tamaños, sectores y regiones geográficas utilizan DynamoDB para crear aplicaciones modernas sin servidor que puedan empezar con pocos recursos y escalarse en todo el mundo. DynamoDB es ideal para casos de uso que requieren un rendimiento uniforme a cualquier escala con una sobrecarga operativa mínima o cero. La siguiente lista incluye algunos casos de uso en los que puede utilizar DynamoDB:

- Aplicaciones de servicios financieros: supongamos que trabaja en una empresa de servicios financieros que crea aplicaciones para el comercio y el enrutamiento en tiempo real, la administración de préstamos, generación de tokens y libros de transacciones. Con las tablas globales de DynamoDB, sus aplicaciones pueden responder a los eventos y atender el tráfico de las Regiones de AWS de su elección con un rendimiento de lectura y escritura rápido y local.

  DynamoDB es adecuado para aplicaciones con los requisitos de disponibilidad más estrictos. Elimina la carga operativa que supone el escalado manual de las instancias para aumentar el almacenamiento o el rendimiento, el control de versiones y las licencias.

  Puede utilizar las transacciones de DynamoDB para lograr atomicidad, coherencia, aislamiento y durabilidad (ACID) en una o más tablas con una sola solicitud. Las transacciones (ACID) se adaptan a las cargas de trabajo que incluyen el procesamiento de transacciones financieras o la gestión de pedidos. DynamoDB se adapta de forma instantánea a sus cargas de trabajo cuando aumentan o disminuyen, lo que le permite escalar su base de datos de forma eficaz para adaptarse a las condiciones del mercado, como los horarios del mercado de valores.

- Aplicaciones de juegos: como empresa de juegos, puede usar DynamoDB para todas las partes de las plataformas de juego como, por ejemplo, el estado del juego, los datos de los jugadores, el historial de sesiones y las tablas de clasificación. Elija DynamoDB por la escalabilidad, rendimiento uniforme y facilidad de las operaciones que ofrece su arquitectura sin servidor. DynamoDB es ideal para las arquitecturas escalables horizontalmente necesarias en las que se basan los juegos de éxito. Reduce y escala horizontalmente de forma rápida el rendimiento del juego (escala a cero sin un arranque en frío). Esta escalabilidad optimiza la eficiencia de su arquitectura, ya sea para escalarla horizontalmente para los picos de tráfico o para reducirla horizontalmente de nuevo cuando el uso del juego sea bajo.
- Aplicaciones de streaming: las empresas de medios y entretenimiento utilizan DynamoDB como índice de metadatos para el contenido, los servicios de administración del contenido o para ofrecer estadísticas deportivas casi en tiempo real. También utilizan DynamoDB para ejecutar servicios de listas de control y marcadores de los usuarios y procesar miles de millones de eventos diarios de los clientes para generar recomendaciones. Estos clientes se benefician de la escalabilidad, el rendimiento y la resiliencia de DynamoDB. DynamoDB se adapta a los cambios en la carga de trabajo a medida que aumentan o disminuyen, lo que facilita los casos de uso con contenido multimedia en streaming que pueden tener cualquier nivel de demanda.

Para obtener más información sobre cómo utilizan DynamoDB clientes de distintos sectores, consulte Amazon DynamoDB Customers y This is My Architecture.

Capacidades de DynamoDB

Replicación multiactiva con tablas globales

Las tablas globales proporcionan una replicación multiactiva de los datos en las Regiones de AWS que elija con una disponibilidad del 99,999 %. Las tablas globales ofrecen una solución totalmente administrada para implementar una base de datos multiregión y multiactiva, sin necesidad de crear y mantener su propia solución de replicación. Con las tablas globales, puede especificar las Regiones de AWS en las que desea que estén disponibles las tablas. DynamoDB replica los cambios de datos en curso en todas estas tablas.

Sus aplicaciones distribuidas por todo el mundo pueden acceder a los datos de forma local en las regiones seleccionadas para lograr un rendimiento de lectura y escritura de milisegundos de un solo dígito. Como las tablas globales son multiactivas, no necesita una tabla principal. Esto significa que no se producen conmutaciones por error complicadas o con retardo ni tiempos de inactividad de la base de datos cuando se conmuta por error una aplicación entre regiones.

Transacciones ACID

DynamoDB está diseñado para cargas de trabajo de misión crítica. Admite transacciones (ACID) para aplicaciones que requieren una lógica empresarial compleja. DynamoDB ofrece compatibilidad nativa del servidor con las transacciones, lo que simplifica la experiencia del desarrollador a la hora de realizar cambios coordinados, de todo o nada, en múltiples elementos dentro de las tablas y entre ellas.

Captura de datos de cambios para arquitecturas basadas en eventos

DynamoDB admite el streaming de registros de captura de datos de cambios (CDC) en los elementos casi en tiempo real. Ofrece dos modelos de streaming para CDC: DynamoDB Streams y Kinesis Data Streams para DynamoDB. Cada vez que una aplicación crea, actualiza o elimina elementos en una tabla, Streams registra una secuencia en orden cronológico de cada cambio en los elementos casi en tiempo real. Esto hace que DynamoDB Streams sea ideal para que las aplicaciones con una arquitectura basada en eventos consuman los cambios y actúen en consecuencia.

Índices secundarios

DynamoDB ofrece la opción de crear índices secundarios globales y locales, que permiten consultar los datos de la tabla mediante una clave alternativa. Con estos índices secundarios, puede acceder a los datos con atributos distintos a los de la clave principal, lo que le proporciona la máxima flexibilidad para obtener acceso a los datos.

Integraciones de servicios

DynamoDB se integra ampliamente con varios Servicios de AWS para que pueda obtener más valor de sus datos, eliminar el trabajo pesado indiferenciado y utilizar sus cargas de trabajo a escala. Algunos ejemplos son: AWS CloudFormation, Amazon CloudWatch, Amazon S3, AWS Identity and Access Management (IAM) y AWS Auto Scaling. En las siguientes secciones, se describen algunas de las integraciones de servicios que puede realizar con DynamoDB:

Integraciones sin servidor

Para crear aplicaciones de extremo a extremo sin servidor, DynamoDB se integra de forma nativa con diversos Servicios de AWS sin servidor. Por ejemplo, puede integrar DynamoDB con

AWS Lambda para crear desencadenadores, que son fragmentos de código que responden automáticamente a los eventos en DynamoDB Streams. Con los desencadenadores, puede crear aplicaciones basadas en eventos que reaccionen a las modificaciones de datos en las tablas de DynamoDB. Para optimizar los costos, puede filtrar los eventos que Lambda procesa desde un flujo de DynamoDB.

En la siguiente lista se presentan algunos ejemplos de integraciones sin servidor con DynamoDB:

- AWS AppSync para crear API de GraphQL
- Amazon API Gateway para crear API de REST
- Lambda para la computación sin servidor
- Amazon Kinesis Data Streams para la captura de cambios de datos (CDC)

Importación y exportación de datos a Amazon S3

La integración de DynamoDB con Amazon S3 le permite exportar datos fácilmente a un bucket de Amazon S3 para utilizar análisis y machine learning. DynamoDB admite exportaciones de tablas completas y exportaciones incrementales para exportar datos modificados, actualizados o eliminados en un periodo de tiempo específico. También puede importar datos desde Amazon S3 a una nueva tabla de DynamoDB.

Integración sin ETL

DynamoDB admite la integración sin ETL con Amazon Redshift y el uso de una canalización de ingesta OpenSearch con Amazon DynamoDB. Estas integraciones le permiten ejecutar análisis complejos y utilizar funcionalidades de búsqueda avanzada en los datos de las tablas de DynamoDB. Por ejemplo, puede realizar búsquedas vectoriales y de texto completo, además de búsquedas semánticas en los datos de DynamoDB. Las integraciones sin ETL no afectan a las cargas de trabajo de producción que se ejecutan en DynamoDB.

Almacenamiento en caché

DynamoDB Accelerator (DAX) es un servicio de almacenamiento en caché de alta disponibilidad y totalmente administrado creado para DynamoDB. DAX multiplica el rendimiento hasta por diez (de milisegundos a microsegundos) incluso con millones de solicitudes por segundo. DAX hace todo el trabajo pesado necesario para agregar aceleración en memoria a las tablas de DynamoDB, sin tener que administrar la invalidación de la caché, el llenado de datos o la administración de clústeres.

Seguridad

DynamoDB utiliza IAM para ayudarle a controlar de forma segura el acceso a los recursos de DynamoDB. Con IAM, se pueden administrar de forma centralizada los permisos que controlan qué usuarios de DynamoDB pueden acceder a los recursos. Utilice IAM para controlar quién está autenticado (ha iniciado sesión) y autorizado (tiene permisos) para utilizar recursos. Como DynamoDB utiliza IAM, no se necesitan nombres de usuario ni contraseñas para acceder. Dado que no tiene que administrar ninguna política complicada de rotación de contraseñas, su estrategia de seguridad se simplifica. Con IAM, también puede habilitar un control de acceso detallado para proporcionar autorización de atributo. También puede definir políticas basadas en recursos compatibles con el Analizador de acceso de AWS IAM y Bloqueo de acceso público (BPA) para simplificar la administración de políticas.

Amazon DynamoDB cifra todos los datos en reposo del cliente de manera predeterminada. El cifrado en reposo mejora la seguridad de los datos mediante el uso de claves de cifrado almacenadas en AWS Key Management Service (AWS KMS). Con el cifrado en reposo, puede crear aplicaciones sensibles a la seguridad que necesitan cumplimiento estricto de cifrado y requisitos normativos. Cuando accede a una tabla cifrada, DynamoDB descifra los datos de la tabla de forma transparente. No es necesario que cambie sus aplicaciones o código para utilizar o administrar tablas cifradas. DynamoDB continúa proporcionando la misma latencia en milisegundos de un solo dígito que cabe esperar y todas las consultas de DynamoDB funcionan sin inconvenientes con los datos cifrados.

Puede especificar si DynamoDB debe utilizar una Clave propiedad de AWS (tipo de cifrado predeterminado), una Clave administrada de AWS o una clave administrada por el cliente para cifrar los datos del usuario. El cifrado predeterminado mediante claves KMS propiedad de AWS está disponible sin costo adicional. Para el cifrado del cliente, puede usar el SDK de cifrado de bases de datos de AWS.

DynamoDB también se adhiere a varios estándares de conformidad, como HIPAA, PCI DSS y el RGPD, lo que le permite cumplir los requisitos normativos.

Resiliencia

De forma predeterminada, DynamoDB replica automáticamente los datos en tres zonas de disponibilidad para ofrecer una alta durabilidad y un SLA de disponibilidad del 99,99 %. DynamoDB también ofrece funcionalidades adicionales para ayudarle a alcanzar sus objetivos de continuidad empresarial y recuperación de desastres.

DynamoDB las siguientes características que le ayudan a satisfacer sus necesidades de resiliencia y copia de seguridad de los datos:

Características

- Tablas globales
- Copias de seguridad continuas y recuperación a un momento dado
- Backup y restauración en diferido

Tablas globales

Las tablas globales de DynamoDB permiten un SLA de disponibilidad del 99,999 % y resiliencia multirregional. Esto le ayuda a crear aplicaciones resilientes y a optimizarlas para el objetivo de tiempo de recuperación (RTO) y el objetivo de punto de recuperación (RPO) más bajos. Las tablas globales también se integran con AWS Fault Injection Service(AWS FIS) para realizar experimentos de inyección de errores en las cargas de trabajo de las tablas globales. Por ejemplo, pausar la replicación de una tabla global en cualquier tabla de réplica.

Copias de seguridad continuas y recuperación a un momento dado

Las copias de seguridad continuas le proporcionan un grado de detalle por segundo y la capacidad de iniciar una recuperación a un momento dado. Con la recuperación a un momento dado, puede restaurar una tabla a cualquier momento de los últimos 35 días con una precisión de segundos. Puede establecer el periodo de recuperación en cualquier valor entre 1 y 35 días.

En las copias de seguridad continuas y el inicio de una restauración a un momento dado no se utiliza capacidad aprovisionada. Tampoco afectan al rendimiento ni a la disponibilidad de las aplicaciones.

Backup y restauración en diferido

La copia de seguridad y la restauración bajo demanda le permiten crear copias de seguridad completas de una tabla para conservarlas a largo plazo y archivarlas para cumplir las normativas. Las copias de seguridad no afectan al rendimiento de la tabla, por lo que puede realizar copias de seguridad de tablas de cualquier tamaño. Con la integración de AWS Backup, puede utilizar AWS Backup para programar, copiar, etiquetar y administrar el ciclo de vida de sus copias de seguridad bajo demanda de DynamoDB de forma automática. Con AWS Backup, puede copiar copias de seguridad bajo demanda en todas las cuentas y regiones, y transferir las copias de seguridad antiguas a un almacenamiento en frío para optimizar los costos.

Acceso a DynamoDB

Puede trabajar con DynamoDB mediante la Consola de administración de AWS, la AWS Command Line Interface, NoSQL Workbench para DynamoDB o las API de DynamoDB.

Para obtener más información, consulte Acceso a DynamoDB.

Precios de DynamoDB

DynamoDB cobra por leer, escribir y almacenar datos en sus tablas, además de cualquier característica opcional que decida habilitar. DynamoDB dispone de dos modos de capacidad con sus respectivas opciones de facturación para procesar lecturas y escrituras en sus tablas: bajo demanda y aprovisionado.

DynamoDB también se incluye en el nivel siempre gratuito, que proporciona 25 GB de almacenamiento. El nivel siempre gratuito también incluye 25 unidades de capacidad de escritura (WCU) y 25 unidades de capacidad de lectura (RCU) aprovisionadas, lo que es suficiente para gestionar 200 millones de solicitudes al mes.

Para obtener más información, consulte los precios de Amazon DynamoDB.

Introducción a DynamoDB

Si es la primera vez que usa DynamoDB, le recomendamos que empiece leyendo lo siguiente:

- Introducción a DynamoDB: lo guía por el proceso de configuración de DynamoDB, la creación de tablas de ejemplo y la carga de datos. En este tema también se proporciona información sobre cómo realizar algunas operaciones básicas de bases de datos mediante la Consola de administración de AWS, la AWS CLI, NoSQL Workbench y las API de DynamoDB.
- Componentes básicos de DynamoDB: describe los conceptos básicos de DynamoDB.
- Prácticas recomendadas para el diseño y la arquitectura con DynamoDB: proporciona recomendaciones sobre el diseño de NoSQL, el enfoque Well-Architected de DynamoDB, el diseño de tablas y otras características de DynamoDB. Estas prácticas recomendadas lo ayudan a maximizar el rendimiento y minimizar los costos de rendimiento al trabajar con DynamoDB.

También le recomendamos que revise los siguientes tutoriales, en los que se ofrecen procedimientos completos para familiarizarse con DynamoDB. Puede completar estos tutoriales mediante la característica de nivel siempre gratuito.

- Create and Query a NoSQL Table with Amazon DynamoDB (Creación y consulta de una tabla NoSQL con Amazon DynamoDB
- Build an Application Using a NoSQL Key-Value Data Store (Creación de una aplicación con un almacén de datos de clave-valor NoSQL

Para obtener información sobre los recursos, las herramientas y las estrategias para migrar a DynamoDB, consulte Migración a DynamoDB. Para leer los blogs y documentos técnicos más recientes, consulte los recursos de Amazon DynamoDB.

Introducción a DynamoDB

En las siguientes secciones, obtendrá información sobre cómo conectarse a las tablas de DynamoDB, a crearlas y a administrarlas.

Antes de comenzar, debe familiarizarse con los conceptos básicos de Amazon DynamoDB. Puede obtener información general rápida en ¿Qué es Amazon DynamoDB? y un análisis más detallado en Componentes básicos de Amazon DynamoDB. Luego, continúe con los Requisitos previos.

> **Note**
>
> Cuando se registra en AWS, puede comenzar a utilizar DynamoDB mediante la Capa gratuita de AWS. Si aún no ha excedido los beneficios del nivel gratuito para Amazon DynamoDB, no incurrirá en costos para completar los ejemplos de esta sección. De lo contrario, incurrirá en tarifas de uso de DynamoDB estándar desde el momento en que cree las tablas hasta que las elimine.
>
> Si no desea registrarse para obtener una cuenta del nivel gratuito, puede configurar DynamoDB local (versión descargable) en el equipo. La versión descargable le permite desarrollar y probar aplicaciones localmente sin obtener acceso a una cuenta de AWS o acceder al servicio web de DynamoDB.

Temas

- Recursos de Amazon DynamoDB para usuarios nuevos
- Acceso a DynamoDB
- Requisitos previos
- Configuración de DynamoDB
- Paso 1: crear una tabla en DynamoDB
- Paso 2: escribir datos en una tabla de DynamoDB
- Paso 3: leer datos de una tabla de DynamoDB
- Paso 4: actualizar los datos de una tabla de DynamoDB
- Paso 5: consultar los datos de una tabla de DynamoDB
- Paso 6: (opcional) Eliminar la tabla de DynamoDB para limpiar los recursos

- Continuación del aprendizaje de DynamoDB
- Generación de código de infraestructura para Amazon DynamoDB mediante Consola a código

Recursos de Amazon DynamoDB para usuarios nuevos

Recomendamos a los nuevos usuarios que empiecen por leer las secciones siguientes y que las consulten cuando sea necesario.

- Precios y aspectos resaltados del servicio: en la página de detalles del producto se proporciona información general sobre DynamoDB junto con casos de uso, aspectos resaltados del servicio y los precios.
- Recursos de DynamoDB: la sección de recursos de DynamoDB cuenta con vídeos, tutoriales y recomendaciones que le presentarán el servicio, los conceptos de modelado de datos y las características y capacidades principales.
- Introducción: la sección Introducción a DynamoDB incluye información sobre cómo configurar DynamoDB, crear tablas de muestra y cargar datos. También proporciona información sobre cómo realizar operaciones básicas con bases de datos utilizando la Consola de administración de AWS, la AWS CLI, NoSQL Workbench y las API de DynamoDB.
- Curso de conceptos básicos de DynamoDB: un curso digital gratuito que enseña los fundamentos de DynamoDB, incluido el diseño de tablas, los tipos de datos y las operaciones básicas. Este curso es ideal para desarrolladores que se inician en DynamoDB y las bases de datos NoSQL.
- DynamoDB Nuggets: una colección de tutoriales en vídeo breves y específicos que explican conceptos y características clave de DynamoDB. Estos vídeos breves tratan temas como el modelado de datos, las estrategias de partición, los patrones de acceso y las prácticas recomendadas en un formato fácil de entender.
- Repositorio de muestras de código de DynamoDB: proporciona ejemplos prácticos de código de DynamoDB en varios lenguajes de programación.
- Formación gratuita sobre DynamoDB: AWS ofrece cursos de formación digital gratuitos que tratan los conceptos, las características y las prácticas recomendadas de DynamoDB. La serie de cursos de DynamoDB en profundidad lo ayuda a obtener información sobre cómo diseñar modelos de datos eficientes, implementar claves de partición eficaces y utilizar índices secundarios de forma efectiva.
- NoSQL Workbench para DynamoDB: NoSQL Workbench es una herramienta visual unificada que proporciona características de modelado de datos, visualización de datos y desarrollo de consultas. Lo ayuda a diseñar, crear, consultar y administrar tablas de DynamoDB, lo que la hace

especialmente valiosa para aprender los conceptos de modelado de datos de DynamoDB y probar patrones de acceso.

- Patrones de diseño de DynamoDB: Prácticas recomendadas y ejemplos de modelado de datos que demuestran las prácticas recomendadas para diferentes casos de uso. Incluye ejemplos prácticos y muestras de código para implementar soluciones eficaces.
- Tutoriales prácticos: tutoriales paso a paso en la Consola de administración de AWS que lo guiarán a través de las tareas habituales de DynamoDB, desde la creación de tablas hasta la implementación de consultas complejas.
- Migración a DynamoDB: esta Migración a DynamoDB proporciona información general sobre el proceso, las herramientas y las estrategias para migrar una base de datos a DynamoDB.
- Enfoque de AWS Well-Architected para DynamoDB: the section called “El enfoque Well-Architected de DynamoDB” proporciona las prácticas recomendadas de arquitectura para diseñar y operar aplicaciones fiables, seguras, eficientes y rentables mediante DynamoDB. Incluye principios de diseño, patrones arquitectónicos y orientación operativa alineados con los seis pilares de Marco de Well-Architected.

Prácticas recomendadas adicionales de Amazon DynamoDB para usuarios nuevos

Cuando complete las secciones anteriores, lea estas secciones:

- Capacidad de rendimiento de DynamoDB

  Proporciona información general de los dos modos de rendimiento disponibles para DynamoDB y consideraciones para seleccionar el modo de capacidad adecuado para la aplicación. El modo bajo demanda es la opción de rendimiento predeterminada y recomendada para la mayoría de las cargas de trabajo de DynamoDB.
- Prácticas recomendadas

  Identifique y aborde los problemas para maximizar el rendimiento y minimizar los costos al trabajar con DynamoDB.

Recursos de AWS CLI

Si desea utilizar la interfaz de la línea de comandos (AWS CLI) de AWS, puede utilizar estos documentos, que lo ayudarán a empezar:

- Documentación de AWS CLI

  En esta sección, se proporciona información sobre cómo descargar la AWS CLI, poner a funcionar la AWS CLI en su sistema y proporcionar sus credenciales de AWS.
- Documentación de AWS CLI para DynamoDB

  Este documento independiente abarca todas las AWS CLI para los comandos de DynamoDB, incluida la sintaxis, además de ejemplos.

Recursos de programación

Puede escribir programas de aplicación para utilizar la API de DynamoDB con diversos lenguajes de programación conocidos. Estos son algunos de los recursos:

- Herramientas para Amazon Web Services

  AWS ofrece una serie de kits de desarrollo de software (SDK) con soporte para DynamoDB. Puede codificar para DynamoDB con Java, .NET, PHP, Ruby y otros lenguajes. Estos SDK pueden simplificar en gran medida el desarrollo de la aplicación mediante al formatear de las solicitudes a DynamoDB, analizar de las respuestas y proporcionar lógica de reintento y gestión de errores.
- Referencia de la API de DynamoDB

  Si no desea utilizar los SDK de AWS, puede interactuar con DynamoDB directamente a través de la API de DynamoDB. Este documento trata todas las operaciones de la API de DynamoDB, incluida la sintaxis y los ejemplos. En esta sección, puede encontrar consejos de solución de problemas e información acerca de la creación y autenticación de solicitudes, así como de la gestión de respuestas.

Acceso a DynamoDB

Puede acceder a Amazon DynamoDB mediante la Consola de administración de AWS, la AWS Command Line Interface (AWS CLI) o la API de DynamoDB.

Temas

- Uso de la consola
- Uso de AWS CLI
- Uso de la API

- Uso de NoSQL Workbench para DynamoDB
- Rangos de direcciones IP
- Puntos de conexión de doble pila para el protocolo de Internet versión 6 (IPv6)

Uso de la consola

Puede acceder a la Consola de administración de AWS para Amazon DynamoDB en https://console.aws.amazon.com/dynamodb/home.

A continuación, se muestran algunas de las acciones que puede realizar en la consola de DynamoDB:

- Administrar las tablas: cree, actualice y elimine las tablas. La calculadora de capacidad puede ayudar a estimar los requisitos de capacidad.
- Interactuar con los datos: consulte, agregue, actualice y elimine elementos de las tablas. Administre la configuración del tiempo de vida (TTL).
- Supervisar y analizar: consulte los paneles, supervise y configure las alarmas y analice las métricas y las alertas de las tablas de DynamoDB.
- Optimizar y ampliar: administre índices secundarios, transmisiones, desencadenadores, capacidad reservada y otras características avanzadas para mejorar el uso de DynamoDB.

La consola de DynamoDB proporciona una interfaz completa para administrar los recursos de DynamoDB. Le recomendamos que acceda a la consola e interactúe con ella para obtener más información.

Uso de AWS CLI

Puede usar la AWS Command Line Interface (AWS CLI) para controlar varios servicios de AWS desde la línea de comandos y automatizarlos mediante scripts. Puede usar la AWS CLI para operaciones ad-hoc, como crear una tabla. También puede usarla para incluir operaciones de Amazon DynamoDB en scripts de utilidades.

Para poder utilizar la AWS CLI con DynamoDB, debe obtener un ID de clave de acceso y una clave de acceso secreta. Para obtener más información, consulte Concesión de acceso mediante programación .

Para obtener un listado completo de todos los comandos disponibles para DynamoDB en la AWS CLI, consulte la Referencia de comandos de AWS CLI.

Descarga y configuración de la AWS CLI

La AWS CLI está disponible en http://aws.amazon.com/cli. Se ejecuta en Windows, macOS o Linux. Después de descargar la AWS CLI, siga estos pasos para instalarla y configurarla:

1. Vaya a la Guía del usuario de la AWS Command Line Interface.
2. Siga las instrucciones de Instalación de la AWS CLI y Configuración de la AWS CLI.

Uso de la AWS CLI con DynamoDB

El formato de la línea de comandos se compone de un nombre de operación de DynamoDB seguido de los parámetros de dicha operación. La AWS CLI admite la sintaxis abreviada de los valores de los parámetros, además de JSON.

Por ejemplo, el comando siguiente crea una tabla llamada Music. La clave de partición es Artist y la de ordenación, SongTitle. Para facilitar la legibilidad, los comandos largos de esta sección se dividen en líneas separadas.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --table-class STANDARD
```

Los comandos siguientes añaden nuevos elementos a la tabla. En estos ejemplos se usa una combinación de sintaxis abreviada y JSON.

```
aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me Today"},
 "AlbumTitle": {"S": "Somewhat Famous"}}' \
    --return-consumed-capacity TOTAL
```

```
aws dynamodb put-item \
    --table-name Music \
    --item '{
        "Artist": {"S": "Acme Band"},
        "SongTitle": {"S": "Happy Day"},
        "AlbumTitle": {"S": "Songs About Life"} }' \
    --return-consumed-capacity TOTAL
```

Puede ser difícil crear código JSON válido en la línea de comandos. Sin embargo, la AWS CLI puede leer archivos JSON. Por ejemplo, fíjese en el fragmento de código JSON siguiente, que se almacena en un archivo denominado key-conditions.json.

```
{
    "Artist": {
        "AttributeValueList": [
            {
                "S": "No One You Know"
            }
        ],
        "ComparisonOperator": "EQ"
    },
    "SongTitle": {
        "AttributeValueList": [
            {
                "S": "Call Me Today"
            }
        ],
        "ComparisonOperator": "EQ"
    }
}
```

Ahora puede emitir una solicitud de `Query` con la AWS CLI. En este ejemplo, el contenido del archivo key-conditions.json se usa para el parámetro `--key-conditions`.

```
aws dynamodb query --table-name Music --key-conditions file://key-conditions.json
```

Uso de la AWS CLI con DynamoDB local

La AWS CLI también puede interactuar con la versión local de DynamoDB (descargable) que se ejecuta en su equipo. Para ello, agregue el parámetro siguiente a cada comando:

```
--endpoint-url http://localhost:8000
```

En e siguiente ejemplo se usa la AWS CLI para mostrar las tablas de una base de datos local:

```
aws dynamodb list-tables --endpoint-url http://localhost:8000
```

Si DynamoDB utiliza un número de transferencia distinto del predeterminado (8000), modifique el valor de `--endpoint-url` en consecuencia.

> **Note**
>
> La AWS CLI no puede utilizar la versión local de DynamoDB (descargable) como punto de conexión predeterminado. Por lo tanto, debe especificar `--endpoint-url` con cada comando.

Uso de la API

Puede utilizar la Consola de administración de AWS y la AWS Command Line Interface para trabajar de manera interactiva con Amazon DynamoDB. Sin embargo, para sacar el máximo partido de DynamoDB, puede escribir el código de aplicación mediante los SDK de AWS.

Los SDK de AWS ofrecen una amplia compatibilidad con DynamoDB en Java, JavaScript en el navegador, .NET, Node.js, PHP, Python, Ruby, C++, Go, Android e iOS.

Para poder utilizar los SDK de AWS con DynamoDB, debe obtener un ID de clave de acceso de AWS y una clave de acceso secreta. Para obtener más información, consulte Configuración de DynamoDB (servicio web) .

Para obtener información general sobre la programación de aplicaciones de DynamoDB con los SDK de AWS, consulte Programación con DynamoDB y los SDK de AWS.

Uso de NoSQL Workbench para DynamoDB

También puede acceder a DynamoDB por medio de la descarga y el uso de NoSQL Workbench para DynamoDB.

NoSQL Workbench para Amazon DynamoDB es una aplicación GUI de cliente multiplataforma que puede usar para el desarrollo moderno de bases de datos y operaciones. Está disponible

para Windows, macOS y Linux. NoSQL Workbench es una herramienta de desarrollo visual que proporciona características de modelado de datos, visualización de datos y desarrollo de consultas para ayudarle a diseñar, crear, consultar y administrar tablas de DynamoDB. NoSQL Workbench ahora incluye DynamoDB local como parte opcional del proceso de instalación, lo que facilita el modelado de los datos en DynamoDB local. Para obtener más información sobre DynamoDB local y sus requisitos, consulte Configuración de la versión de DynamoDB local (versión descargable) .

> **Note**
>
> NoSQL Workbench para DynamoDB actualmente no admite inicios de sesión de AWS configurados con la autenticación de dos factores (2FA).

Modelado de datos

Con NoSQL Workbench para DynamoDB, puede crear nuevos modelos de datos o diseñar modelos basados en modelos de datos existentes que satisfagan los patrones de acceso a datos de su aplicación. También puede importar y exportar el modelo de datos diseñado al final del proceso. Para obtener más información, consulte Creación de modelos de datos con NoSQL Workbench.

Visualización de datos

El visualizador de modelos de datos proporciona un lienzo donde puede mapear las consultas y visualizar los patrones de acceso (facetas) de la aplicación sin tener que escribir código. Cada faceta corresponde a un patrón de acceso diferente en DynamoDB. Puede generar automáticamente datos de muestra para utilizarlos en su modelo de datos. Para obtener más información, consulte Visualización de patrones de acceso a datos.

Generación de operaciones

NoSQL Workbench proporciona una completa interfaz gráfica de usuario rica para que desarrolle y pruebe las consultas. Puede utilizar el generador de operaciones para visualizar, explorar y consultar conjuntos de datos en directo. También puede utilizar el generador de operaciones estructurado para crear y realizar operaciones de plano de datos. Admite expresiones de proyección y condición, y le permite generar código de muestra en varios idiomas. Para obtener más información, consulte Exploración de conjuntos de datos y creación de operaciones con NoSQL Workbench.

Rangos de direcciones IP

Amazon Web Services (AWS) publica sus rangos de direcciones IP actuales en formato JSON. Para ver los rangos actuales, descargue ip-ranges.json. Para obtener más información, consulte Rangos de direcciones IP de AWS en Referencia general de AWS.

Para conocer los rangos de direcciones IP que se pueden utilizar para acceder a tablas e índices de DynamoDB, busque la siguiente cadena en el archivo ip-ranges.json: `"service": "DYNAMODB"`.

> **Note**
>
> Los rangos de direcciones IP no se aplican a DynamoDB Streams ni a DynamoDB Accelerator (DAX).

Puntos de conexión de doble pila para el protocolo de Internet versión 6 (IPv6)

DynamoDB ofrece puntos de conexión de doble pila compatibles con IPv4 e IPv6. Las convenciones de nomenclatura de los puntos de conexión son:

- `dynamodb.<region>.api.aws`
- `<account-id>.ddb.<region>.api.aws`
- `streams-dynamodb.<region>.api.aws`
- `dax.<region>.api.aws`
- `dynamodb-fips.<region>.api.aws`

Para obtener una lista completa de los puntos de conexión y disponibilidad regional de DynamoDB, consulte Puntos de conexión y cuotas de Amazon DynamoDB en la Guía de referencia general de AWS.

Para obtener más información sobre la configuración de la CLI de AWS para usar puntos de conexión de doble pila, consulte la sección Configurar el uso de puntos de conexión de doble pila para todos los servicios de AWS en la Guía de interfaz de línea de comandos de AWS.

Para obtener más información sobre la configuración de los clientes del SDK para que utilicen puntos de conexión de doble pila, consulte el tema sobre Puntos de conexión de doble pila y FIPS en la Guía de herramientas y AWS SDK.

Antes de usar DynamoDB con IPv6, debe actualizar el rol de usuario de IAM o las políticas basadas en recursos que usa para el filtrado de direcciones IP para incluir los rangos de direcciones IPv6. Las políticas de filtrado de direcciones IP que no tienen en cuenta la dirección IPv6 pueden provocar problemas de acceso. Para obtener más información, consulte Operadores de condición de dirección IP en la Guía de AWS Identity and Access Management.

Requisitos previos

Antes de iniciar el tutorial de Amazon DynamoDB, obtenga información sobre las formas de acceder a DynamoDB en Acceso a DynamoDB. A continuación, configure DynamoDB mediante el servicio web o la versión descargada localmente en Configuración de DynamoDB . Después de eso, continúe hasta Paso 1: crear una tabla en DynamoDB.

> **Note**
>
> - Si solo va a interactuar con DynamoDB a través de la Consola de administración de AWS, no necesita una clave de acceso de AWS. Realice los pasos que se indican en Inscripción en AWS y, a continuación, siga por el Paso 1: crear una tabla en DynamoDB.
> - Si no desea registrarse para obtener una cuenta del nivel gratuito, puede configurar DynamoDB local (versión descargable). A continuación, siga por el Paso 1: crear una tabla en DynamoDB.
> - Existen diferencias a la hora de trabajar con comandos de la CLI en terminales de Linux y Windows. En la siguiente guía se presentan comandos formateados para terminales de Linux (esto incluye macOS) y comandos formateados para CMD de Windows. Elija el comando que mejor se adapte a la aplicación de terminal que esté utilizando.

Configuración de DynamoDB

Además del servicio web Amazon DynamoDB, AWS ofrece una versión descargable de DynamoDB que se puede ejecutar localmente en el equipo. La versión descargable es útil para desarrollar y probar su código. Le permite crear y probar aplicaciones localmente sin obtener acceso al servicio web de DynamoDB.

En los temas de esta sección se describe cómo configurar DynamoDB (versión descargable) y el servicio web de DynamoDB.

Temas

- Configuración de DynamoDB (servicio web)
- Configuración de la versión de DynamoDB local (versión descargable)

Configuración de DynamoDB (servicio web)

Para utilizar el servicio web de Amazon DynamoDB:

1. Regístrese en AWS.
2. Obtenga una clave de acceso de AWS (usada para obtener acceso a DynamoDB mediante programación).

   > **Note**
   >
   > Si solo va a interaccionar con DynamoDB a través de la Consola de administración de AWS, no se requiere la clave de acceso de AWS, en cuyo caso puede ir directamente a Uso de la consola.

3. Configure sus credenciales (utilizadas para obtener acceso a DynamoDB).

Inscripción en AWS

Para utilizar el servicio de DynamoDB, debe disponer de una cuenta de AWS. Si aún no tiene una, se le pedirá que la cree cuando se inscriba. No se le cobrará por los servicios de AWS en los que se inscriba, salvo si los utiliza.

Para inscribirse en AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga las instrucciones que se le indiquen.

   Parte del procedimiento de registro consiste en recibir una llamada telefónica o mensaje de texto e indicar un código de verificación en el teclado del teléfono.

   Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario y utilice únicamente el usuario raíz para realizar tareas que requieren acceso de usuario raíz.

Concesión de acceso mediante programación

Para poder acceder a DynamoDB mediante programación o a través de la AWS Command Line Interface (AWS CLI), debe disponer de acceso mediante programación. No necesita el acceso mediante programación si piensa utilizar solamente la consola de DynamoDB.

Los usuarios necesitan acceso programático si desean interactuar con AWS fuera de la Consola de administración de AWS. La forma de conceder el acceso programático depende del tipo de usuario que acceda a AWS.

Para conceder acceso programático a los usuarios, seleccione una de las siguientes opciones.

¿Qué usuario necesita acceso programático?	Para	Mediante
IAM	(Recomendado) Use credenciales de consola como credenciales temporales para firmar las solicitudes programáticas a la AWS CLI, los AWS SDK y las API de AWS.	Siga las instrucciones de la interfaz que desea utilizar: • Para la AWS CLI, consulte Iniciar sesión para el desarrollo local de AWS en la Guía del usuario de AWS Command Line Interface. • Para los AWS SDK, consulte Iniciar sesión para el desarrollo local de AWS en la Guía de referencia de AWS SDK y herramientas.
Identidad del personal (Usuarios administrados en el IAM Identity Center)	Utiliza credenciales temporales para firmar las solicitudes programáticas a la AWS CLI, los AWS SDK y las API de AWS.	Siga las instrucciones de la interfaz que desea utilizar: • Para utilizar la AWS CLI, consulte Configuring the AWS CLI to usa AWS IAM Identity Center en la Guía del usuario de AWS Command Line Interface.

¿Qué usuario necesita acceso programático?	Para	Mediante
		• Para usar AWS SDK, las herramientas y las API de AWS, consulta IAM Identity Center authentication en la Guía de referencia del SDK y las herramientas de AWS.
IAM	Utiliza credenciales temporales para firmar las solicitudes programáticas a la AWS CLI, los AWS SDK y las API de AWS.	Siguiendo las instrucciones de Uso de credenciales temporales con recursos de AWS de la Guía del usuario de IAM.
IAM	(No recomendado) Utilizar credenciales a largo plazo para firmar las solicitudes programáticas a la AWS CLI, los AWS SDK o las API de AWS.	Siga las instrucciones de la interfaz que desea utilizar: • Para la AWS CLI, consulte Autenticación mediante credenciales de usuario de IAM en la Guía del usuario de AWS Command Line Interface. • Para ver los AWS SDK y las herramientas, consulte Autenticar mediante credenciales a largo plazo en la Guía de referencia de AWS SDK y herramientas. • Para las API de AWS, consulta Administración de claves de acceso para usuarios de IAM en la Guía del usuario de IAM.

Configuración de las credenciales

Para poder acceder a DynamoDB mediante programación o a través de la AWS CLI, debe configurar sus credenciales para habilitar la autorización para sus aplicaciones.

Puede hacer esto de varias formas. Por ejemplo, puede crear manualmente el archivo de credenciales para almacenar el ID de clave de acceso y la clave de acceso secreta. También puede utilizar el comando `aws configure` de la AWS CLI para crear automáticamente el archivo. Otra opción consiste en usar variables de entorno. Para obtener más información sobre cómo configurar sus credenciales, consulte la guía para desarrolladores del SDK de AWS específico de su lenguaje de programación.

Para instalar y configurar la AWS CLI, consulte Uso de AWS CLI.

Integración con otros servicios de DynamoDB

Puede integrar DynamoDB con muchos otros servicios de AWS. Para obtener más información, consulte los siguientes temas:

- Uso de DynamoDB con otros servicios de AWS
- CloudFormation para DynamoDB
- Uso de AWS Backup con DynamoDB
- AWS Identity and Access Management (IAM) y DynamoDB
- Uso de AWS Lambda con Amazon DynamoDB

Configuración de la versión de DynamoDB local (versión descargable)

Con la versión descargable de Amazon DynamoDB puede desarrollar y probar aplicaciones sin obtener acceso al servicio web de DynamoDB. En lugar de ello, se utiliza una base de datos autónoma que reside en el equipo. Cuando esté listo para implementar la aplicación en producción, elimine el punto de enlace local en el código y, a continuación, apunte al servicio web de DynamoDB.

Disponer de esta versión local supone un ahorro en cuanto a rendimiento, almacenamiento de datos y tarifas de transferencia de datos. Además, no es necesario disponer de una conexión a Internet mientras se desarrolla la aplicación.

DynamoDB local está disponible como descarga (requiere JRE), como una dependencia de Apache Maven o como una imagen de Docker.

Si prefiere utilizar el servicio web de Amazon DynamoDB en su lugar, consulte la Configuración de DynamoDB (servicio web) .

Temas

- Implementación de DynamoDB localmente en la computadora
- Notas sobre el uso local de DynamoDB
- Historial de versiones de DynamoDB local
- Telemetría en DynamoDB local

Implementación de DynamoDB localmente en la computadora

> **Note**
>
> - DynamoDB local está disponible en tres versiones: v3.x (actual), v2.x (heredada) y v1.x (obsoleta).
> - Se recomienda DynamoDB v3.x para su uso local de pruebas y desarrollo.
> - La migración de DynamoDB local V2.x a V3.x requiere actualizar las instrucciones de importación de `com.amazonaws.services.dynamodbv2` a `software.amazon.dynamodb` y actualizar las dependencias de Maven para los usuarios de Maven.
> - Si va a migrar una aplicación que usa el SDK para Java v1.x al SDK para Java 2.x, siga los pasos para el SDK de AWS para Java 2.x.

Descarga de DynamoDB local

Siga estos pasos para configurar y ejecutar DynamoDB en el ordenador.

Para configurar DynamoDB en su ordenador

1. Descargue DynamoDB local gratis desde una de las siguientes ubicaciones.

Enlaces de descarga	Sumas de comprobación
.tar.gz | .zip	.tar.gz.sha256 | .zip.sha256

> **Important**
>
> Para ejecutar la versión 2.6.0 o posterior de DynamoDB en el equipo, debe disponer del Entorno de ejecución de Java (JRE) versión 17.x o posteriores. La aplicación no se ejecuta en versiones anteriores de JRE.

2. Después de descargar el archivo, extraiga el contenido y copie el directorio extraído en la ubicación que prefiera.
3. Para iniciar DynamoDB en el ordenador, abra una ventana del símbolo del sistema, vaya al directorio donde ha extraído `DynamoDBLocal.jar` e ingrese el comando siguiente.

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb
```

> **Note**
>
> Si utiliza Windows PowerShell, asegúrese de delimitar el nombre del parámetro o todo el nombre y el valor del siguiente modo:
> `java -D"java.library.path=./DynamoDBLocal_lib" -jar DynamoDBLocal.jar`
> DynamoDB procesa las solicitudes entrantes hasta que lo detiene. Para detener DynamoDB, escriba Ctrl+C en la ventana del símbolo del sistema.
> De manera predeterminada, DynamoDB usa el puerto 8000. Si el puerto 8000 no está disponible, este comando genera una excepción. Para obtener una lista completa de opciones de tiempo de ejecución de DynamoDB, incluida `-port`, ingrese este comando.
> `java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -help`

4. Para poder acceder a DynamoDB mediante programación o a través de la AWS Command Line Interface (AWS CLI), debe configurar sus credenciales para habilitar la autorización para sus aplicaciones. DynamoDB descargable requiere cualquier credencial para funcionar, como se muestra en el siguiente ejemplo.

```
AWS Access Key ID: "fakeMyKeyId"
AWS Secret Access Key: "fakeSecretAccessKey"
Default Region Name: "fakeRegion"
```

Puede utilizar el comando `aws configure` de la AWS CLI para configurar las credenciales. Para obtener más información, consulte Uso de AWS CLI.

5. Comience a escribir aplicaciones. Para acceder a DynamoDB ejecutado localmente con la AWS CLI, use el parámetro `--endpoint-url` . Por ejemplo, utilice el siguiente comando para enumerar las tablas de DynamoDB.

```
aws dynamodb list-tables --endpoint-url http://localhost:8000
```

Ejecución de DynamoDB local como imagen de Docker

La versión descargable de Amazon DynamoDB también está disponible como imagen de Docker. Para obtener más información, consulte dynamodb-local. Para ver la versión local actual de DynamoDB, ejecute el siguiente comando:

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -version
```

Para ver un ejemplo del uso de DynamoDB local como parte de una aplicación REST construida en AWS Serverless Application Model (AWS SAM), consulte Aplicación DynamoDB de SAM para administración de pedidos. Esta aplicación de ejemplo demuestra cómo utilizar DynamoDB local para pruebas.

Si desea ejecutar una aplicación multicontenedor que también utilice el contenedor local de DynamoDB, utilice Docker Compose para definir y ejecutar todos los servicios de la aplicación, incluido DynamoDB local.

Para instalar y ejecutar DynamoDB local con Docker Compose:

1. Descargue e instale Docker Desktop.
2. Copie el siguiente código en un archivo y guárdelo como `docker-compose.yml`.

```
services:
 dynamodb-local:
   command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
   image: "amazon/dynamodb-local:latest"
   container_name: dynamodb-local
   ports:
     - "8000:8000"
```

```
    volumes:
      - "./docker/dynamodb:/home/dynamodblocal/data"
    working_dir: /home/dynamodblocal
```

Si desea que su aplicación y DynamoDB local estén en contenedores independientes use el siguiente archivo yaml.

```
version: '3.8'
services:
 dynamodb-local:
   command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
   image: "amazon/dynamodb-local:latest"
   container_name: dynamodb-local
   ports:
     - "8000:8000"
   volumes:
     - "./docker/dynamodb:/home/dynamodblocal/data"
   working_dir: /home/dynamodblocal
 app-node:
   depends_on:
     - dynamodb-local
   image: amazon/aws-cli
   container_name: app-node
   ports:
    - "8080:8080"
   environment:
     AWS_ACCESS_KEY_ID: 'DUMMYIDEXAMPLE'
     AWS_SECRET_ACCESS_KEY: 'DUMMYEXAMPLEKEY'
   command:
     dynamodb describe-limits --endpoint-url http://dynamodb-local:8000 --region
 us-west-2
```

Este script docker-compose.yml crea un contenedor `app-node` y un contenedor `dynamodb-local`. El script ejecuta un comando en el contenedor `app-node` que utiliza la AWS CLI para conectarse con el contenedor `dynamodb-local` y describe los límites de cuenta y tabla.

Para usar con su propia imagen de aplicación, reemplace el valor `image` del ejemplo siguiente por el de su aplicación.

```
version: '3.8'
services:
```

```
dynamodb-local:
  command: "-jar DynamoDBLocal.jar -sharedDb -dbPath ./data"
  image: "amazon/dynamodb-local:latest"
  container_name: dynamodb-local
  ports:
    - "8000:8000"
  volumes:
    - "./docker/dynamodb:/home/dynamodblocal/data"
  working_dir: /home/dynamodblocal
app-node:
  image: location-of-your-dynamodb-demo-app:latest
  container_name: app-node
  ports:
    - "8080:8080"
  depends_on:
    - "dynamodb-local"
  links:
    - "dynamodb-local"
  environment:
    AWS_ACCESS_KEY_ID: 'DUMMYIDEXAMPLE'
    AWS_SECRET_ACCESS_KEY: 'DUMMYEXAMPLEKEY'
    REGION: 'eu-west-1'
```

> **Note**
>
> Los scripts de YAML requieren que especifique una clave de acceso de AWS y una clave secreta de AWS, pero no se requiere que sean claves de AWS válidas para que pueda acceder a DynamoDB local.

3. Ejecute el siguiente comando de línea de comandos:

```
docker-compose up
```

Ejecución de DynamoDB local como dependencia de Apache Maven

> **Note**
>
> Si va a migrar una aplicación que usa el SDK para Java v1.x al SDK para Java 2.x, siga los pasos para el SDK de AWS para Java 2.x.

Siga estos pasos para usar Amazon DynamoDB en su aplicación como dependencia.

Para implementar DynamoDB como repositorio de Apache Maven

1. Descargue e instale Apache Maven. Para obtener más información, consulte Downloading Apache Maven e Installing Apache Maven.
2. Agregue el repositorio de Maven para DynamoDB al archivo POM (Project Object Model) de la aplicación.

```
<!--Dependency:-->
<dependencies>
   <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>DynamoDBLocal</artifactId>
      <version>3.1.0</version>
   </dependency>
</dependencies>
```

Ejemplo de plantilla para usar con Spring Boot 3 o Spring Framework 6:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>org.example</groupId>
<artifactId>SpringMavenDynamoDB</artifactId>
<version>1.0-SNAPSHOT</version>

<properties>
   <spring-boot.version>3.0.1</spring-boot.version>
   <maven.compiler.source>17</maven.compiler.source>
   <maven.compiler.target>17</maven.compiler.target>
   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

   <parent>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-parent</artifactId>
       <version>3.0.1</version>
```

```
    </parent>

<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>DynamoDBLocal</artifactId>
        <version>3.1.0</version>
    </dependency>
    <!-- Spring Boot -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
        <version>${spring-boot.version}</version>
    </dependency>
    <!-- Spring Web -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>${spring-boot.version}</version>
    </dependency>
    <!-- Spring Data JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
        <version>${spring-boot.version}</version>
    </dependency>
    <!-- Other Spring dependencies -->
    <!-- Replace the version numbers with the desired version -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>6.0.0</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>6.0.0</version>
    </dependency>
    <!-- Add other Spring dependencies as needed -->
    <!-- Add any other dependencies your project requires -->
</dependencies>
</project>
```

> **Note**
>
> También puede usar la URL del repositorio central de Maven.

Ejecución de DynamoDB local en AWS CloudShell

AWS CloudShell es un intérprete de comandos previamente autenticado y basado en navegador que se puede lanzar directamente desde la Consola de administración de AWS. Puede navegar a AWS CloudShell desde la Consola de administración de AWS de varias maneras diferentes. Para obtener más información, consulta Introducción a AWS CloudShell.

Siga estos pasos para ejecutar DynamoDB local en AWS CloudShell en cualquier lugar de la Consola de administración de AWS.

Ejecución de DynamoDB local en AWS CloudShell en la Consola de administración de AWS

1. Inicie AWS CloudShell desde la interfaz de la consola, seleccione una Región de AWS disponible y cambie al intérprete de comandos que prefiera, por ejemplo, Bash, PowerShell o Z shell.
2. Para elegir una Región de AWS, vaya al menú Seleccionar una región y seleccione una Región de AWS compatible en la que trabajar. (Las regiones disponibles aparecen resaltadas).
3. Desde la Consola de administración de AWS, puede iniciar AWS CloudShell mediante la elección de una de las siguientes opciones:
   a. En la barra de navegación, elija el icono AWS CloudShell.
   b. En el cuadro Buscar, introduzca la palabra “CloudShell” y, a continuación, seleccione CloudShell.
   c. En el widget Visitas recientes, seleccione CloudShell.
   d. En la barra de herramientas de la consola, elija CloudShell.
4. Para ejecutar DynamoDB local en AWS CloudShell puede utilizar el alias `dynamodb-local`. Puede especificar opciones adicionales de la línea de comandos para cambiar la configuración de DynamoDB local. Consulte the section called “Notas de uso” para conocer las opciones disponibles.

> **Note**
>
> Para ejecutar DynamoDB local en segundo plano, ejecute DynamoDB local en AWS CloudShell mediante: `dynamodb-local &`.

5. Para acceder a DynamoDB ejecutado localmente en AWS CloudShell con la AWS CLI, use el parámetro `--endpoint-url`. Por ejemplo, utilice el siguiente comando para enumerar las tablas de DynamoDB:

   ```
   aws dynamodb list-tables --endpoint-url http://localhost:8000
   ```

Para ver un ejemplo de un proyecto de ejemplo que muestre varios enfoques para configurar y usar DynamoDB local, incluida la descarga de archivos JAR, su ejecución como imagen de Docker y su uso como una dependencia de Maven, consulte DynamoDB Local Sample Java Project.

Notas sobre el uso local de DynamoDB

Excepto en lo tocante al punto de enlace, las aplicaciones que se ejecutan con la versión descargable de Amazon DynamoDB en el sistema deberían funcionar también con el servicio web de DynamoDB. Sin embargo, si utiliza DynamoDB localmente, debe tener en cuenta lo siguiente:

- Si utiliza la opción `-sharedDb`, DynamoDB crea un único archivo de base de datos denominado shared-local-instance.db. Todos los programas que se conectan con DynamoDB obtienen acceso a este archivo. Si elimina el archivo, perderá todos los datos que haya guardado en él.
- Si omite `-sharedDb`, el archivo de base de datos recibirá el nombre myaccesskeyid_region.db, con el ID de clave de acceso de AWS y la región de AWS que aparecen en la configuración de la aplicación. Si elimina el archivo, perderá todos los datos que haya guardado en él.
- Si usa la opción `-inMemory`, DynamoDB no escribe ningún archivo de base de datos. En lugar de ello, todos los datos se escriben en la memoria y ninguno de ellos se guarda cuando cierra DynamoDB.
- Si utiliza la opción `-inMemory`, también será necesaria la opción `-sharedDb`.
- Si utiliza la opción `-optimizeDbBeforeStartup`, también debe especificar el parámetro `-dbPath` para que DynamoDB pueda encontrar el archivo de base de datos.
- Los SDK de AWS para DynamoDB requieren que se especifiquen en la configuración de la aplicación un valor de clave de acceso y un valor de región de AWS. A no ser que utilice la opción `-sharedDb` o `-inMemory`, DynamoDB usará estos valores para asignar el nombre al archivo de

base de datos local. Estos valores no tienen que ser valores de AWS válidos para la ejecución local. Sin embargo, tal vez le convenga utilizar valores válidos para que pueda ejecutar el código en la nube más adelante cambiando simplemente el punto de enlace que esté utilizando.

- DynamoDB local siempre devuelve un valor nulo para `billingModeSummary`.
- `AWS_ACCESS_KEY_ID` de DynamoDB local solo puede contener letras (A-Z, a-z) y números (0-9).
- DynamoDB local no admite la recuperación en un momento dado (PITR).

Temas

- Opciones de línea de comandos
- Configuración del punto de conexión local
- Diferencias entre la versión descargable de DynamoDB y el servicio web de DynamoDB

Opciones de línea de comandos

Puede usar las siguientes opciones de línea de comandos con la versión descargable de DynamoDB:

- `-cors value`: habilita la compatibilidad con el uso compartido de recursos entre orígenes (CORS, por sus siglas en inglés) para JavaScript. Debe proporcionar una lista de dominios específicos "permitidos" separados por comas. El ajuste predeterminado para `-cors` es el asterisco (*), que permite el acceso público.
- `-dbPath value`: directorio donde DynamoDB escribe el archivo de base de datos. Si no especifica esta opción, el archivo se escribe en el directorio actual. Puede especificar tanto `-dbPath` como `-inMemory` a la vez.
- `-delayTransientStatuses`: hace que DynamoDB presente retardos para algunas operaciones. DynamoDB (versión descargable) puede realizar algunas tareas casi instantáneamente, como operaciones de creación, actualización y eliminación en tablas e índices. Sin embargo, el servicio de DynamoDB requiere más tiempo para estas tareas. El establecimiento de este parámetro ayuda a que DynamoDB ejecutándose en su ordenador simule mejor el comportamiento del servicio web de DynamoDB. En la actualidad, este parámetro introduce retardos solo para los índices secundarios globales que se encuentran en el estado CREATING o DELETING.
- `-help`: imprime un resumen de uso y las opciones posibles.

- `-inMemory`: DynamoDB se ejecuta en memoria, en lugar de usar un archivo de base de datos. Cuando detenga DynamoDB, no se guardará ninguno de los datos. Puede especificar tanto `-dbPath` como `-inMemory` a la vez.
- `-optimizeDbBeforeStartup`: optimiza las tablas de la base de datos subyacente antes de iniciar DynamoDB en el ordenador. Si utiliza este parámetro, también debe especificar `-dbPath`.
- `-port value`: número de puerto que DynamoDB utiliza para comunicarse con la aplicación. Si no especifica esta opción, el puerto predeterminado es `8000`.

> **Note**
>
> De manera predeterminada, DynamoDB usa el puerto 8000. Si el puerto 8000 no está disponible, este comando genera una excepción. Puede usar la opción `-port` para especificar otro número de puerto. Para obtener una lista completa de opciones de tiempo de ejecución de DynamoDB, incluida `-port`, escriba este comando: `java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -help`

- `-sharedDb`: si especifica `-sharedDb`, DynamoDB utiliza un solo archivo de base de datos, en lugar de archivos distintos para cada credencial y región.
- `-disableTelemetry`: si se especifica, DynamoDB local no enviará ninguna telemetría.
- `-version`: imprime la versión de DynamoDB local.

Configuración del punto de conexión local

De forma predeterminada, los SDK y las herramientas de AWS utilizan los puntos de enlace del servicio web de Amazon DynamoDB. Para utilizar los SDK y las herramientas con la versión descargable de DynamoDB, debe especificar el punto de enlace local:

`http://localhost:8000`

AWS Command Line Interface

Puede utilizar la AWS Command Line Interface (AWS CLI) para interactuar con la versión de DynamoDB descargable.

Para acceder a la instancia de DynamoDB que se ejecuta localmente, use el parámetro `--endpoint-url`. A continuación se muestra un ejemplo de cómo usar la AWS CLI para obtener una lista de las tablas de DynamoDB en el ordenador.

```
aws dynamodb list-tables --endpoint-url http://localhost:8000
```

> **Note**
>
> La AWS CLI no puede utilizar la versión descargable de DynamoDB como punto de enlace predeterminado. Por lo tanto, debe especificar `--endpoint-url` con cada comando de la AWS CLI.

AWS SDK

La forma de especificar el punto de enlace depende del lenguaje de programación y del SDK de AWS que se utilizan en cada caso. En las secciones siguientes se describe cómo hacerlo:

- Java: configuración de la región y del punto de conexión de AWS (DynamoDB local admite el SDK para Java de AWS V1 y V2)
- CodeSamples.Java.RegionAndEndpoint .NET: configuración de la región y del punto de conexión de AWS

Diferencias entre la versión descargable de DynamoDB y el servicio web de DynamoDB

La versión descargable de DynamoDB se ha desarrollado únicamente para tareas de desarrollo y comprobación. Por el contrario, el servicio web de DynamoDB es un servicio administrado con características de escalabilidad, disponibilidad y durabilidad que resulta ideal para utilizarlo en producción.

La versión descargable de DynamoDB se diferencia del servicio web en lo siguiente:

- En el cliente no se admiten las Regiones de AWS ni determinadas Cuentas de AWS.
- Los ajustes de rendimiento aprovisionado se omiten en la versión descargable de DynamoDB, aunque la operación `CreateTable` los requiera. Para `CreateTable`, puede especificar cualquier cifra que desee de desempeño provisionado de lectura y escritura, si bien estas cifras no se utilizarán. Puede llamar a `UpdateTable` tantas veces como desee al día. Sin embargo, se omiten los cambios en los valores de desempeño provisionado.
- `Scan`Las operaciones se llevan a cabo secuencialmente. No se admiten los exámenes en paralelo. Los parámetros `Segment` y `TotalSegments` de la operación `Scan` se pasan por alto.

- La velocidad de las operaciones de lectura y escritura en los datos de la tabla solamente se ve limitada por la velocidad del equipo. Las operaciones `CreateTable`, `UpdateTable` y `DeleteTable` se llevan a cabo de inmediato y el estado de la tabla siempre es ACTIVE. Las operaciones `UpdateTable` que solo cambian los ajustes de desempeño provisionado de las tablas y/o los índices secundarios globales se realizan de inmediato. Si una operación `UpdateTable` crea o elimina los índices secundarios globales, estos índices pasan sucesivamente por los estados normales (tales como CREATING o DELETING, respectivamente) antes de entrar en el estado ACTIVE. La tabla permanece en el estado ACTIVE durante este tiempo.
- Las operaciones de lectura son de consistencia final. No obstante, debido a la velocidad de ejecución local de DynamoDB en la computadora, la mayoría de las lecturas parecen tener una alta coherencia.
- No se realiza el seguimiento de las métricas ni de los tamaños de las colecciones de elementos. En las respuestas a las operaciones, se devuelven valores Null en lugar de las métricas de las colecciones de elementos.
- En DynamoDB, existe un límite de 1 MB para los datos devueltos en cada conjunto de resultados. Tanto el servicio web de DynamoDB como la versión descargable imponen este límite. Sin embargo, al consultar un índice, el servicio de DynamoDB solo calcula el tamaño de la clave y los atributos previstos. En cambio, la versión descargable de DynamoDB calcula el tamaño del elemento completo.
- Si utiliza DynamoDB Streams, la velocidad a la que se crean las particiones puede ser diferente. En el servicio web de DynamoDB, el comportamiento de creación de particiones depende en parte de la actividad de partición de la tabla. Cuando ejecuta DynamoDB localmente, la tabla no se particiona. La aplicación no debe depender del comportamiento de los fragmentos, puesto que son efímeros en ambos casos.
- La versión descargable de DynamoDB no lanza `TransactionConflictExceptions` para API transaccionales. Le recomendamos que utilice una plataforma de simulación de Java para simular `TransactionConflictExceptions` en el controlador de DynamoDB y probar cómo responde la aplicación a las transacciones conflictivas.
- En el servicio web de DynamoDB, tanto si se accede a través de la consola como de la AWS CLI, en los nombres de las tablas se distingue entre mayúsculas y minúsculas. Una tabla llamada `Authors` y otra llamada `authors` pueden existir como tablas independientes. En la versión descargable, los nombres de las tablas no distinguen entre mayúsculas y minúsculas y si intenta crear estas dos tablas se producirá un error.
- La versión descargable de DynamoDB no admite el etiquetado.

- La versión descargable de DynamoDB ignora el parámetro Limit de ExecuteStatement.

Historial de versiones de DynamoDB local

En la siguiente tabla se describen los cambios importantes de cada versión de DynamoDB local.

Versión	Cambio	Descripción	Fecha
3.1.0	Mejora del rendimiento de las consultas PartiQL, incluida la dependencia de Joda-Time	• Actualización de la versión Java del SDK a la última versión pública, de 2.25.50 a 2.33.0 • Incluida la dependencia de Joda-time en el archivo Pom.xml • Mejora del rendimiento de las consultas PartiQL • Actualización de las dependencias para corregir varios problemas de vulnerabilidad de CVE	14 de septiembre de 2025
3.0.0	Migración de AWS SDK Java V1 a V2	• Migración de AWS SDK Java V1 a V2 • Estructura de paquetes actualizada de `com.amazonaws.services.dynamodbv2` a `software.`	17 de julio de 2025

Versión	Cambio	Descripción	Fecha
		`amazon.dynamodb.services` • Se han eliminado las dependencias del AWS SDK Java V1	
2.6.0	Compatibilidad con el ARN de tabla como nombre de tabla en las API de DynamoDB Corrección del rendimiento y actualizaciones de seguridad	• Se agregó compatibilidad para usar el ARN de tabla como nombre de tabla en varias API de DynamoDB • Corrección del error `CreateStreamTable` en máquinas de alto rendimiento, como Mac M3 • Actualización de las dependencias para corregir los problemas de vulnerabilidad (CVE-2022-49043,CVE-2024-56732, CVE-2020-29582, CVE-2025-21502, CVE-2024-50602, CVE-2025-24970, CVE-2025-25193)	13 de marzo de 2025

Versión	Cambio	Descripción	Fecha
2.5.4	Actualización a Jetty Dependencies	• Actualización de Jetty 12.0.8 a Jetty 12.0.14 (resuelve CVE-2024-6763, CVE-2024-8184, CVE-2024-47535) Mitigación para (CVE-2024-21634)	12 de diciembre de 2024
2.5.3	Actualización de las dependencias de Jackson a 2.17.x en Log4j Core (se resuelve CVE-2022-1471)	• Actualización de las dependencias de Jackson a 2.17.x en Log4j Core (se resuelve CVE-2022-1471) para solucionar una vulnerabilidad de seguridad crítica en la biblioteca SnakeYAML, que es una dependencia transitiva	6 de noviembre de 2024

Versión	Cambio	Descripción	Fecha
2.5.2	Corrección de errores en el flujo de trabajo de actualización de tabla	• Corrección de errores en el flujo de trabajo cuando la actualización de la tabla intenta actualizar la tabla con el modo de facturación bajo demanda a aprovisionado con GSI	20 de junio de 2024
2.5.1	Revisión de errores introducidos en la característica `OndemandThroughPut`	• Se han corregido un par de errores relacionados con `OndemandThroughPut`	5 de junio de 2024
2.5.0	Compatibilidad con un rendimiento máximo configurable para las tablas bajo demanda `ReturnValuesOnConditionCheckFailure`, `BatchExecuteStatement` y `ExecuteTransaction` `Request`.	• Adición de telemetría al modo incrustado • Corrección de la traducción de SDKv2 para `ConditionalCheckException`	28 de mayo de 2024

Versión	Cambio	Descripción	Fecha
2.4.0	Compatibilidad de `ReturnValuesOnConditionCheckFailure` : modo integrado	• Corrección de modo integrado para `TrimmedDataAccessException` para el funcionamiento en varias secuencias • Corrección de la traducción de excepciones para el SDKv2 en el modo integrado	17 de abril de 2024
2.3.0	Actualización de Jetty y JDK	• Actualización a Jetty 12.0.2 • Actualización a JDK 17 • Actualización de ANTLR4 a 4.10.1	14 de marzo de 2024
2.2.0	Se ha añadido compatibilidad con la protección contra la eliminación de tablas y el parámetro `ReturnValuesOnConditionCheckFailure`	• Se ha añadido compatibilidad con la protección contra la eliminación de tablas • Se ha añadido compatibilidad con ReturnValuesOnConditionCheckFailure • Se ha añadido compatibilidad con el indicador -version	14 de diciembre de 2023

Versión	Cambio	Descripción	Fecha
2.1.0	Compatibilidad para bibliotecas nativas de SQLLite para proyectos de Maven y agregar telemetría	• Agregar telemetría a DynamoDB local • Copiar dinámicamente las bibliotecas nativas de SQLLite para proyectos de Maven • Se eliminó la biblioteca io.github.ganadist.sqlite4java de la dependencia de Maven • Actualización de GoogleGuava a 32.1.1-jre	23 de octubre de 2023
2.0.0	Migración de javax al espacio de nombres de Jakarta y compatibilidad de JDK11	• Migración de javax al espacio de nombres de Jakarta y compatibilidad de JDK11 • Solución para gestionar el acceso no válido y la clave secreta mientras el servidor startup • Solucionar las vulnerabilidades identificadas por Maven mediante la actualización de las dependencias	5 de julio de 2023

Versión	Cambio	Descripción	Fecha
1.25.1	Actualización de las dependencias de Jackson a 2.17.x en Log4j Core (se resuelve CVE-2022-1471)	Actualización de las dependencias de Jackson a 2.17.x en Log4j Core (se resuelve CVE-2022-1471) para solucionar una vulnerabilidad de seguridad crítica en la biblioteca SnakeYAML, que es una dependencia transitiva	6 de noviembre de 2024
1.25.0	Se ha añadido compatibilidad con la protección contra la eliminación de tablas y el parámetro `ReturnValuesOnConditionCheckFailure`	• Se ha añadido compatibilidad con la protección contra la eliminación de tablas • Se ha añadido compatibilidad con ReturnValuesOnConditionCheckFailure • Se ha añadido compatibilidad con el indicador -version	18 de diciembre de 2023

Versión	Cambio	Descripción	Fecha
1.24.0	Compatibilidad para bibliotecas nativas de SQLLite para proyectos de Maven y agregar telemetría	• Agregar telemetría a DynamoDB local • Copiar dinámicamente las bibliotecas nativas de SQLLite para proyectos de Maven • Se eliminó la biblioteca io.github.ganadist.sqlite4java de la dependencia de Maven • Actualización de GoogleGuava a 32.1.1-jre	23 de octubre de 2023
1.23.0	Gestionar el acceso no válido y la clave secreta mientras el servidor startup	• Solución para gestionar el acceso no válido y la clave secreta mientras el servidor startup • Solucionar las vulnerabilidades identificadas por Maven mediante la actualización de las dependencias	28 de junio de 2023

Versión	Cambio	Descripción	Fecha
1.22.0	Compatibilidad con Limit Operation para PartiQL	• Optimizar la cláusula IN para PartiQL • Compatibilidad con Limit Operation • Compatibilidad de M1 para proyectos de Maven	8 de junio de 2023
1.21.0	Compatibilidad para 100 acciones por transacción	• Se incrementaron las acciones por transacción de 25 a 100 • Actualización de la imagen de Docker Open JDK a 11 • Se corrige la paridad de la excepción que se produce cuando se duplican elementos en BatchExec uteStatement	26 de enero de 2023
1.20.0	Se ha agregado la compatibilidad para M1 Mac	• Se ha agregado la compatibilidad para M1 Mac • Actualización de la dependencia de Jetty a 9.4.48.v2 0220622	12 de septiembre de 2022

Versión	Cambio	Descripción	Fecha
1.19.0	Se ha actualizado el analizador PartiQL	Se ha actualizado el analizador PartiQL y otras bibliotecas relacionadas	27 de julio de 2022
1.18.0	Se ha actualizado log4j-core y Jackson-core	Se ha actualizado log4j-core a 2.17.1 y Jackson-core 2.10.x a 2.12.0	10 de enero de 2022
1.17.2	Se ha actualizado log4j-core	Se ha actualizado la dependencia de log4j-core a la versión 2.16	16 de enero de 2021
1.17.1	Se ha actualizado log4j-core	Se ha actualizado la dependencia log4j-core para corregir una vulnerabilidad de día cero y evitar la ejecución de código remoto - Log4Shel	10 de enero de 2021
1.17.0	Shell web de Javascript obsoleto	• Se ha actualizado la dependencia del AWS SDK al AWS SDK para Java 1.12.x • Shell web de Javascript obsoleto	8 de enero de 2021

Telemetría en DynamoDB local

En AWS, desarrollamos y lanzamos servicios en función de lo que aprendemos de las interacciones con los clientes y utilizamos el feedback de los clientes para retocar nuestros productos. La

telemetría es información adicional que nos ayuda a comprender mejor nuestras necesidades de cliente, diagnosticar problemas y ofrecer características que mejoren la experiencia de cliente.

DynamoDB local recopila datos de telemetría, como métricas de uso genéricas, información de sistemas y entornos y errores. Para obtener más información sobre los tipos de telemetría recopilados, consulte Tipos de información recopilada.

DynamoDB local no recopila información personal, como nombres de usuario o direcciones de correo electrónico. Tampoco extrae información confidencial en el nivel de proyecto.

Como cliente, puede controlar si la telemetría está activada y puede cambiar la configuración en cualquier momento. Si la telemetría permanece activada, DynamoDB local envía los datos de telemetría en segundo plano sin requerir ninguna interacción adicional con el cliente.

Desactivar la telemetría mediante las opciones de la línea de comandos

Puede desactivar la telemetría mediante las opciones de la línea de comandos al iniciar DynamoDB local mediante esta opción `-disableTelemetry`. Para obtener más información, consulte Opciones de línea de comandos .

Desactivar la telemetría para una sola sesión

En los sistemas operativos macOS y Linux, puede desactivar la telemetría para una sola sesión. Para desactivar la telemetría de la sesión actual, ejecute el siguiente comando para establecer la variable de entorno `DDB_LOCAL_TELEMETRY` en `false`. Repita el comando para cada nuevo terminal o sesión.

```
export DDB_LOCAL_TELEMETRY=0
```

Desactivar la telemetría del perfil en todas las sesiones

Ejecute los siguientes comandos para desactivar la telemetría en todas las sesiones cuando ejecute DynamoDB local en el sistema operativo.

Para desactivar la telemetría en Linux

1. Ejecute:

   ```
   echo "export DDB_LOCAL_TELEMETRY=0" >>~/.profile
   ```

2. Ejecute:

```
source ~/.profile
```

Para desactivar la telemetría en macOS

1. Ejecute:

```
echo "export DDB_LOCAL_TELEMETRY=0" >>~/.profile
```

2. Ejecute:

```
source ~/.profile
```

Para desactivar la telemetría en Windows

1. Ejecute:

```
setx DDB_LOCAL_TELEMETRY 0
```

2. Ejecute:

```
refreshenv
```

Desactive la telemetría mediante el DynamoDB local incrustado en proyectos de Maven

Puede desactivar la telemetría mediante el DynamoDB local incrustado en proyectos de Maven

```
boolean disableTelemetry = true;
// AWS SDK v1
 AmazonDynamoDB amazonDynamoDB =
 DynamoDBEmbedded.create(disableTelemetry).amazonDynamoDB();

// AWS SDK v2
DynamoDbClient ddbClientSDKv2Local =
 DynamoDBEmbedded.create(disableTelemetry).dynamoDbClient();
```

Tipos de información recopilada

- Información de uso: la telemetría genérica, como el inicio o la parada del servidor y la API o la operación a la que se llamó.
- Información del sistema y del entorno: la versión de Java, el sistema operativo (Windows, Linux o macOS), el entorno en el que se ejecuta DynamoDB local (por ejemplo, JAR independiente, contenedor de Docker o como dependencia de Maven) y valores hash de los atributos de uso.

Más información

Los datos de la telemetría que recopila DynamoDB local cumplen con las políticas de privacidad de datos de AWS. Para más información, consulte los siguientes temas:

- Condiciones del servicio de AWS
- Preguntas frecuentes sobre privacidad de datos

Paso 1: crear una tabla en DynamoDB

En este paso, se crea una tabla en Amazon DynamoDB denominada `Music`. La tabla tiene las características siguientes:

- Clave de partición: `Artist`
- Clave de clasificación: `SongTitle`

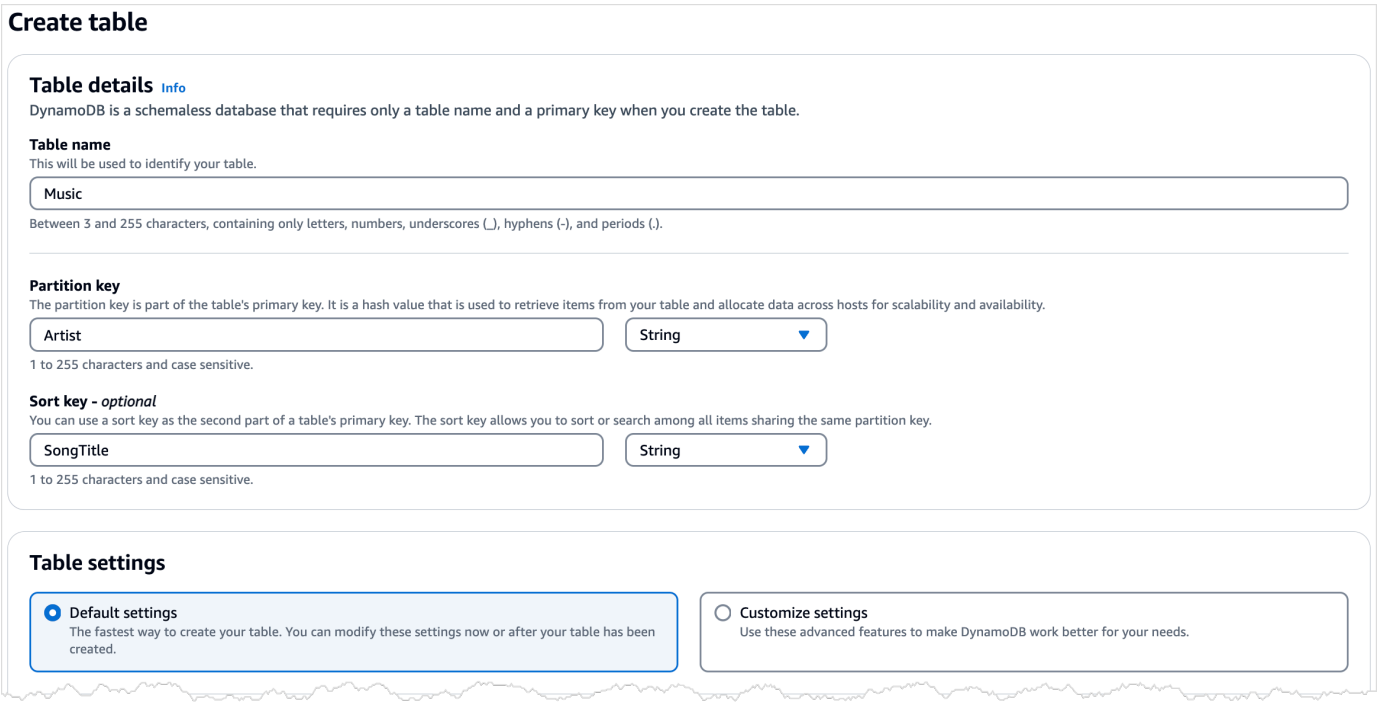
Para obtener más información sobre las operaciones con tablas, consulte Uso de tablas y datos en DynamoDB.

> **Note**
>
> Antes de comenzar, asegúrese de que ha realizado los pasos que se detallan en Requisitos previos.

Consola de administración de AWS

Para crear una tabla denominada `Music` mediante la consola de DynamoDB:

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación izquierdo, elija Tables (Tablas).
3. Seleccione Create table (Creación de tabla).
4. Ingrese los Detalles de la tabla de la siguiente manera:
   a. En Nombre de la tabla, introduzca **`Music`**.
   b. En Partition key (Clave de partición), ingrese **`Artist`**.
   c. Para Clave de clasificación, ingrese **`SongTitle`**.
5. Para Configuración de tabla, mantenga la selección predeterminada de Configuración predeterminada.
6. Elija Creación de tabla para crear la tabla.



7. Una vez que la tabla esté en estado `ACTIVE`, recomendamos que habilite Copias de seguridad a un momento dado para DynamoDB en la tabla realizando los siguientes pasos:
   a. Elija el nombre de la tabla para abrirla.
   b. Elija Backups.
   c. Elija Edición en la sección de Recuperación en un momento dado (PITR).
   d. En la página Edición de configuración de recuperación en un momento dado, elija Activación de recuperación en un momento dado.

e. Seleccione Save changes (Guardar cambios).

AWS CLI

En el siguiente ejemplo de la AWS CLI, se crea una tabla denominada `Music` con `create-table`.

Linux

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --table-class STANDARD
```

CMD de Windows

```
aws dynamodb create-table ^
    --table-name Music ^
    --attribute-definitions ^
        AttributeName=Artist,AttributeType=S ^
        AttributeName=SongTitle,AttributeType=S ^
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE ^
    --billing-mode PAY_PER_REQUEST ^
    --table-class STANDARD
```

El uso de `create-table` devuelve el siguiente resultado de ejemplo.

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
```

```
                "AttributeType": "S"
            }
        ],
        "TableName": "Music",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-03-29T12:11:43.379000-04:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:111122223333:table/Music",
        "TableId": "60abf404-1839-4917-a89b-a8b0ab2a1b87",
        "TableClassSummary": {
            "TableClass": "STANDARD"
        }
    }
}
}
```

Observe que el valor del campo `TableStatus` es `CREATING`.

Para comprobar que DynamoDB ha terminado de crear la tabla `Music`, utilice el comando `describe-table`.

Linux

```
aws dynamodb describe-table --table-name Music | grep TableStatus
```

CMD de Windows

```
aws dynamodb describe-table --table-name Music | findstr TableStatus
```

Este comando devuelve el siguiente resultado. Cuando DynamoDB termina de crear la tabla, el valor del campo `TableStatus` se establece en `ACTIVE`.

```
"TableStatus": "ACTIVE",
```

Una vez que la tabla esté en estado `ACTIVE`, se considera recomendable habilitar Copias de seguridad a un momento dado para DynamoDB en la tabla mediante la ejecución del siguiente comando:

Linux

```
aws dynamodb update-continuous-backups \
    --table-name Music \
    --point-in-time-recovery-specification \
        PointInTimeRecoveryEnabled=true
```

CMD de Windows

```
aws dynamodb update-continuous-backups --table-name Music --point-in-time-recovery-specification PointInTimeRecoveryEnabled=true
```

Este comando devuelve el siguiente resultado.

```
{
    "ContinuousBackupsDescription": {
        "ContinuousBackupsStatus": "ENABLED",
        "PointInTimeRecoveryDescription": {
            "PointInTimeRecoveryStatus": "ENABLED",
            "EarliestRestorableDateTime": "2023-03-29T12:18:19-04:00",
            "LatestRestorableDateTime": "2023-03-29T12:18:19-04:00"
        }
    }
}
```

> **Note**
>
> La habilitación de las copias de seguridad continuas con recuperación a un momento dado puede afectar al costo. Para obtener más información sobre los precios, consulte Precios de Amazon DynamoDB.

AWS SDK

En los siguientes ejemplos de código se muestra cómo crear una tabla de DynamoDB con un SDK de AWS.

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
 CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
```

```
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
```

```
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
{tableName}. {ex.Message}");
            throw;
        }
    }
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table with on-demand billing."
    echo " -n table_name  -- The name of the table to create."
    echo " -a attribute_definitions -- JSON file path of a list of attributes and
 their types."
    echo " -k key_schema -- JSON file path of a list of attributes and their key
 types."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:a:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      a) attribute_definitions="${OPTARG}" ;;
      k) key_schema="${OPTARG}" ;;
      h)
        usage
        return 0
```

```
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_definitions" ]]; then
    errecho "ERROR: You must provide an attribute definitions json file path the
-a parameter."
    usage
    return 1
  fi

  if [[ -z "$key_schema" ]]; then
    errecho "ERROR: You must provide a key schema json file path the -k
parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    attribute_definitions:   $attribute_definitions"
  iecho "    key_schema:   $key_schema"
  iecho ""

  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --billing-mode PAY_PER_REQUEST \
    --key-schema file://"$key_schema" )

  local error_code=${?}
```

```
  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi

  return 0
}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
```

```
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información sobre la API, consulte CreateTable en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Create an Amazon DynamoDB table.
/*!
  \sa createTable()
  \param tableName: Name for the DynamoDB table.
  \param primaryKey: Primary key for the DynamoDB table.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::createTable(const Aws::String &tableName,
                                   const Aws::String &primaryKey,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::cout << "Creating table " << tableName <<
              " with a simple primary key: \"" << primaryKey << "\"." <<
 std::endl;

    Aws::DynamoDB::Model::CreateTableRequest request;

    Aws::DynamoDB::Model::AttributeDefinition hashKey;
    hashKey.SetAttributeName(primaryKey);
    hashKey.SetAttributeType(Aws::DynamoDB::Model::ScalarAttributeType::S);
    request.AddAttributeDefinitions(hashKey);

    Aws::DynamoDB::Model::KeySchemaElement keySchemaElement;
    keySchemaElement.WithAttributeName(primaryKey).WithKeyType(
            Aws::DynamoDB::Model::KeyType::HASH);
    request.AddKeySchema(keySchemaElement);

    Aws::DynamoDB::Model::ProvisionedThroughput throughput;
    throughput.WithReadCapacityUnits(5).WithWriteCapacityUnits(5);
    request.SetProvisionedThroughput(throughput);
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::CreateTableOutcome &outcome =
 dynamoClient.CreateTable(
              request);
    if (outcome.IsSuccess()) {
        std::cout << "Table \""
                  << outcome.GetResult().GetTableDescription().GetTableName() <<
                  " created!" << std::endl;
    }
```

```
    else {
        std::cerr << "Failed to create table: " <<
 outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Código que espera a que se active la tabla.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
```

```
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obtener información sobre la API, consulte CreateTable en la Referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Ejemplo 1: Creación de una tabla con etiquetas

En el siguiente ejemplo `create-table`, se utilizan los atributos y el esquema de claves especificados para crear una tabla denominada `MusicCollection`. Esta tabla utiliza el rendimiento aprovisionado y se cifrará en reposo con la CMK predeterminada propiedad de AWS. El comando también aplica una etiqueta a la tabla, con una clave de `Owner` y un valor de `blueTeam`.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

Salida:

```
{
```

```
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "CREATING",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": "2020-05-26T16:04:41.627000-07:00",
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    }
}
```

Para obtener más información, consulte Operaciones básicas con tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 2: Creación de una tabla en modo bajo demanda

En el siguiente ejemplo, se crea una tabla denominada `MusicCollection` mediante el modo bajo demanda, en lugar del modo de rendimiento aprovisionado. Esto resulta útil para tablas con cargas de trabajo impredecibles.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:44:10.807000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
```

```
            "WriteCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        }
    }
}
```

Para obtener más información, consulte Operaciones básicas con tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 3: creación de una tabla y cifrarla con una CMK administrada por el cliente

En el siguiente ejemplo, se crea una tabla denominada `MusicCollection` y se cifra mediante una CMK administrada por el cliente.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
```

```
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:12:16.431000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "SSEDescription": {
            "Status": "ENABLED",
            "SSEType": "KMS",
            "KMSMasterKeyArn": "arn:aws:kms:us-west-2:123456789012:key/abcd1234-
abcd-1234-a123-ab1234a1b234"
        }
    }
}
```

Para obtener más información, consulte Operaciones básicas con tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 4: creación de una tabla con un índice secundario local

En el siguiente ejemplo, se utilizan los atributos y el esquema de claves especificados para crear una tabla denominada `MusicCollection` con un índice secundario local denominado `AlbumTitleIndex`.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\": \"Artist\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\": \"AlbumTitle\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\": \"INCLUDE\",
                    \"NonKeyAttributes\": [\"Genre\", \"Year\"]
                }
            }
        ]"
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
```

```
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Genre",
                        "Year"
                    ]
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
```

```
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ]
    }
}
```

Para obtener más información, consulte Operaciones básicas con tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 5: creación de una tabla con un índice secundario global

En el siguiente ejemplo, se crea una tabla llamada `GameScores` con un índice secundario global denominado `GameTitleIndex`. La tabla base tiene una clave de partición de `UserId` y una clave de ordenación de `GameTitle`, lo que le permite encontrar eficientemente la mejor puntuación de un usuario individual para un juego específico, mientras que el GSI tiene una clave de partición de `GameTitle` y una clave de ordenación de `TopScore`, lo que te permite encontrar rápidamente la puntuación más alta en general para un juego en particular.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
```

```
                }
            }
        ]"
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T17:28:15.602000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
```

```
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            }
        ]
    }
}
```

Para obtener más información, consulte Operaciones básicas con tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 6: creación de una tabla con varios índices secundarios globales a la vez

En el siguiente ejemplo, se crea una tabla denominada `GameScores` con dos índices secundarios globales. Los esquemas GSI se transfieren mediante un archivo, en lugar de hacerlo a través de la línea de comandos.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes file://gsi.json
```

Contenido de `gsi.json`:

```
[
    {
        "IndexName": "GameTitleIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "TopScore",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        }
    },
    {
        "IndexName": "GameDateIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "Date",
                "KeyType": "RANGE"
```

```
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        }
    }
]
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Date",
                "AttributeType": "S"
            },
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
```

```
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-08-04T16:40:55.524000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            },
            {
                "IndexName": "GameDateIndex",
                "KeySchema": [
                    {
```

```
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "Date",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameDateIndex"
            }
        ]
    }
}
```

Para obtener más información, consulte Operaciones básicas con tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 7: creación de una tabla que tiene habilitado Streams

En el siguiente ejemplo, se crea una tabla denominada `GameScores` con DynamoDB Streams habilitado. En el flujo se escribirán tanto las imágenes nuevas como las antiguas de cada elemento.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
```

```
    --stream-specification StreamEnabled=TRUE,StreamViewType=NEW_AND_OLD_IMAGES
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T10:49:34.056000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_AND_OLD_IMAGES"
        },
        "LatestStreamLabel": "2020-05-27T17:49:34.056",
```

```
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2020-05-27T17:49:34.056"
    }
}
```

Para obtener más información, consulte Operaciones básicas con tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 8: creación de una tabla con un flujo habilitado solo de claves

En el siguiente ejemplo, se crea una tabla denominada `GameScores` con DynamoDB Streams habilitado. Solo se escriben en el flujo los atributos de clave del elementos modificados.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=KEYS_ONLY
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
```

```
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:45:34.140000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "KEYS_ONLY"
        },
        "LatestStreamLabel": "2023-05-25T18:45:34.140",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2023-05-25T18:45:34.140",
        "DeletionProtectionEnabled": false
    }
}
```

Para obtener más información, consulte Captura de datos de cambios para DynamoDB Streams en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 9: creación de una tabla mediante la clase de tabla de acceso poco frecuente estándar de DynamoDB

En el siguiente ejemplo se crea una tabla denominada `GameScores` y asigna la clase de tabla Estándar - Acceso poco frecuente (DynamoDB Standard-IA). Esta clase de tabla está optimizada para que el almacenamiento sea el costo dominante.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
```

```
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:33:07.581000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "TableClassSummary": {
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
```

```
        },
        "DeletionProtectionEnabled": false
    }
}
```

Para obtener más información, consulte Clases de tabla en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 10: creación de una tabla con la protección contra eliminación habilitada

En el siguiente ejemplo, se crea una tabla denominada `GameScores` y habilita la protección contra eliminación.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --deletion-protection-enabled
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
```

```
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T23:02:17.093000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "DeletionProtectionEnabled": true
    }
}
```

Para obtener más información, consulte Uso de la protección contra eliminación en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte CreateTable en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
```

```
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
```

```
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.OnDemandThroughput;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateTable {
    public static void main(String[] args) {
        final String usage = """

            Usage:
                <tableName> <key>

            Where:
                tableName - The Amazon DynamoDB table to create (for example,
 Music3).
                key - The key for the Amazon DynamoDB table (for example,
 Artist).
            """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        System.out.println("Creating an Amazon DynamoDB table " + tableName + "
 with a simple primary key: " + key);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();
```

```
        String result = createTable(ddb, tableName, key);
        System.out.println("New table is " + result);
        ddb.close();
    }

    public static String createTable(DynamoDbClient ddb, String tableName, String
 key) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        CreateTableRequest request = CreateTableRequest.builder()
            .attributeDefinitions(AttributeDefinition.builder()
                .attributeName(key)
                .attributeType(ScalarAttributeType.S)
                .build())
            .keySchema(KeySchemaElement.builder()
                .attributeName(key)
                .keyType(KeyType.HASH)
                .build())
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
 scales based on traffic.
            .tableName(tableName)
            .build();

        String newTable;
        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
 dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            newTable = response.tableDescription().tableName();
            return newTable;

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    BillingMode: "PAY_PER_REQUEST",
  });
```

```
  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
```

```
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun createNewTable(
    tableNameVal: String,
    key: String,
): String? {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef)
            keySchema = listOf(keySchemaVal)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        var tableArn: String
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        tableArn = response.tableDescription!!.tableArn.toString()
        println("Table $tableArn is ready")
        return tableArn
    }
}
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear una tabla.

```
        $tableName = "ddb_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

    public function createTable(string $tableName, array $attributes)
    {
        $keySchema = [];
        $attributeDefinitions = [];
        foreach ($attributes as $attribute) {
            if (is_a($attribute, DynamoDBAttribute::class)) {
                $keySchema[] = ['AttributeName' => $attribute->AttributeName,
'KeyType' => $attribute->KeyType];
                $attributeDefinitions[] =
                    ['AttributeName' => $attribute->AttributeName,
'AttributeType' => $attribute->AttributeType];
            }
        }

        $this->dynamoDbClient->createTable([
            'TableName' => $tableName,
            'KeySchema' => $keySchema,
            'AttributeDefinitions' => $attributeDefinitions,
            'ProvisionedThroughput' => ['ReadCapacityUnits' => 10,
'WriteCapacityUnits' => 10],
        ]);
    }
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para PHP.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: en este ejemplo se crea una tabla denominada Thread que tiene una clave principal compuesta por ForumName (hash de tipos de clave) y Subject (rango de tipos de clave). El esquema utilizado para construir la tabla se puede canalizar hacia cada cmdlet tal como se muestra o se especifica con el parámetro -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Salida:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Ejemplo 2: en este ejemplo se crea una tabla denominada Thread que tiene una clave principal compuesta por ForumName (hash de tipo clave) y Subject (rango de tipos de clave). También se define un índice secundario local. La clave del índice secundario local se establecerá automáticamente a partir de la clave hash principal de la tabla (ForumName). El esquema utilizado para construir la tabla se puede canalizar hacia cada cmdlet tal como se muestra o se especifica con el parámetro -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
```

```
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Salida:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

Ejemplo 3: en este ejemplo se muestra cómo usar una sola canalización para crear una tabla denominada Thread que tiene una clave principal compuesta por ForumName (hash de tipo clave) y Subject (rango de tipos de clave), además de un índice secundario local. Add-DDBKeySchema y Add-DDBIndexSchema crean un nuevo objeto TableSchema para usted si no se proporciona ninguno desde la canalización o el parámetro -Schema.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Salida:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
```

```
LocalSecondaryIndexes  : {LastPostIndex}
```

- Para obtener detalles de la API, consulte CreateTable en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: en este ejemplo se crea una tabla denominada Thread que tiene una clave principal compuesta por ForumName (hash de tipos de clave) y Subject (rango de tipos de clave). El esquema utilizado para construir la tabla se puede canalizar hacia cada cmdlet tal como se muestra o se especifica con el parámetro -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Salida:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Ejemplo 2: en este ejemplo se crea una tabla denominada Thread que tiene una clave principal compuesta por ForumName (hash de tipo clave) y Subject (rango de tipos de clave). También se define un índice secundario local. La clave del índice secundario local se establecerá automáticamente a partir de la clave hash principal de la tabla (ForumName). El esquema utilizado para construir la tabla se puede canalizar hacia cada cmdlet tal como se muestra o se especifica con el parámetro -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
```

```
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Salida:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

Ejemplo 3: en este ejemplo se muestra cómo usar una sola canalización para crear una tabla denominada Thread que tiene una clave principal compuesta por ForumName (hash de tipo clave) y Subject (rango de tipos de clave), además de un índice secundario local. Add-DDBKeySchema y Add-DDBIndexSchema crean un nuevo objeto TableSchema para usted si no se proporciona ninguno desde la canalización o el parámetro -Schema.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Salida:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- Para obtener detalles de la API, consulte CreateTable en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Cree una tabla para almacenar datos de películas.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
```

```
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                BillingMode='PAY_PER_REQUEST',
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Creates an Amazon DynamoDB table that can be used to store movie data.
  # The table uses the release year of the movie as the partition key and the
  # title as the sort key.
  #
  # @param table_name [String] The name of the table to create.
  # @return [Aws::DynamoDB::Table] The newly created table.
  def create_table(table_name)
    @table = @dynamo_resource.create_table(
      table_name: table_name,
      key_schema: [
        { attribute_name: 'year', key_type: 'HASH' }, # Partition key
        { attribute_name: 'title', key_type: 'RANGE' } # Sort key
      ],
      attribute_definitions: [
        { attribute_name: 'year', attribute_type: 'N' },
```

```
        { attribute_name: 'title', attribute_type: 'S' }
      ],
      billing_mode: 'PAY_PER_REQUEST'
    )
    @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
    @table
  rescue Aws::DynamoDB::Errors::ServiceError => e
    @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
    raise
  end
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
pub async fn create_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<CreateTableOutput, Error> {
    let a_name: String = key.into();
    let table_name: String = table.into();

    let ad = AttributeDefinition::builder()
        .attribute_name(&a_name)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .map_err(Error::BuildError)?;

    let ks = KeySchemaElement::builder()
        .attribute_name(&a_name)
```

```
        .key_type(KeyType::Hash)
        .build()
        .map_err(Error::BuildError)?;

    let create_table_response = client
        .create_table()
        .table_name(table_name)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .send()
        .await;

    match create_table_response {
        Ok(out) => {
            println!("Added table {} with key {}", table, key);
            Ok(out)
        }
        Err(e) => {
            eprintln!("Got an error creating table:");
            eprintln!("{}", e);
            Err(Error::unhandled(e))
        }
    }
}
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    TRY.
```

```
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
        oo_result = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
 returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
        " This exception can happen if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
{ lo_resourceinuseex->av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- Para obtener información sobre la API, consulte CreateTable en la Referencia de la API del AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB


    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = CreateTableInput(
                attributeDefinitions: [
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
 "year", attributeType: .n),
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
 "title", attributeType: .s)
                ],
                billingMode: DynamoDBClientTypes.BillingMode.payPerRequest,
                keySchema: [
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
 keyType: .hash),
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
 keyType: .range)
                ],
                tableName: self.tableName
            )
            let output = try await client.createTable(input: input)
            if output.tableDescription == nil {
                throw MoviesError.TableNotFound
            }
```

```
        } catch {
            print("ERROR: createTable:", dump(error))
            throw error
        }
    }
```

- Para obtener detalles sobre la API, consulte CreateTable en la Referencia de la API del SDK de AWS para Swift.

Para obtener más ejemplos de DynamoDB, consulte Ejemplos de código de DynamoDB con los SDK de AWS.

Después de crear la tabla, continúe en el Paso 2: escribir datos en una tabla de DynamoDB.

Paso 2: escribir datos en una tabla de DynamoDB

En este paso, se agregan varios elementos a la tabla `Music` que creó en el Paso 1: crear una tabla en DynamoDB.

Para obtener más información sobre las operaciones de escritura, consulte Escritura de un elemento.

Consola de administración de AWS

Siga estos pasos para escribir datos en la tabla `Music` mediante la consola de DynamoDB.

1. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación izquierdo, elija Tables (Tablas).
3. En la página Tablas, elija la tabla Música.
4. Elija Explore table items (Explorar elementos de la tabla).
5. En la sección Elementos devueltos, elija Creación de elemento.
6. En la página Creación de elemento, haga lo siguiente para agregar elementos a la tabla:

    a. Elija Add new attribute (Agregar nuevo atributo) y, a continuación, elija Number (Número).
    b. Para nombre de atributo, ingrese **`Awards`**.
    c. Repita este proceso para crear un **`AlbumTitle`** de tipo String (Cadena).
    d. Ingrese los valores siguientes para el elemento:

i. En Artist (Artista), escriba **`No One You Know`**.

ii. En SongTitle, escriba **`Call Me Today`**.

iii. En AlbumTitle, escriba **`Somewhat Famous`**.

iv. En Awards (Premios), escriba **`1`**.

7. Elija Create Item (Crear elemento).
8. Repita este proceso y cree otro elemento con los valores siguientes:

   a. En Artist (Artista), escriba **`Acme Band`**.

   b. En SongTitle escriba **`Happy Day`**.

   c. En AlbumTitle, escriba **`Songs About Life`**.

   d. En Awards (Premios), escriba **`10`**.
9. Haga esto una vez más para crear otro elemento con el mismo Artist (Artista) que el paso anterior, pero valores diferentes para los demás atributos:

   a. En Artist (Artista), escriba **`Acme Band`**.

   b. En SongTitle escriba **`PartiQL Rocks`**.

   c. En AlbumTitle, escriba **`Another Album Title`**.

   d. En Awards (Premios), escriba **`8`**.

AWS CLI

En el siguiente ejemplo de AWS CLI se crean varios elementos nuevos en la tabla mediante `Music`. Puede hacerlo mediante la API de DynamoDB o PartiQL, un lenguaje de consulta compatible con SQL para DynamoDB.

DynamoDB API

Linux

```
aws dynamodb put-item \
    --table-name Music  \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me Today"},
 "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "1"}}'

aws dynamodb put-item \
```

```
    --table-name Music  \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Howdy"},
 "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "2"}}'

aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"},
 "AlbumTitle": {"S": "Songs About Life"}, "Awards": {"N": "10"}}'

aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "PartiQL Rocks"},
 "AlbumTitle": {"S": "Another Album Title"}, "Awards": {"N": "8"}}'
```

CMD de Windows

```
aws dynamodb put-item ^
    --table-name Music  ^
    --item ^
        "{\"Artist\": {\"S\": \"No One You Know\"}, \"SongTitle\": {\"S\": \"Call
 Me Today\"}, \"AlbumTitle\": {\"S\": \"Somewhat Famous\"}, \"Awards\": {\"N\":
 \"1\"}}"

aws dynamodb put-item ^
    --table-name Music  ^
    --item ^
        "{\"Artist\": {\"S\": \"No One You Know\"}, \"SongTitle\": {\"S\": \"Howdy
\"}, \"AlbumTitle\": {\"S\": \"Somewhat Famous\"}, \"Awards\": {\"N\": \"2\"}}"

aws dynamodb put-item ^
    --table-name Music ^
    --item ^
        "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day\"},
 \"AlbumTitle\": {\"S\": \"Songs About Life\"}, \"Awards\": {\"N\": \"10\"}}"

aws dynamodb put-item ^
    --table-name Music ^
    --item ^
        "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"PartiQL Rocks
\"}, \"AlbumTitle\": {\"S\": \"Another Album Title\"}, \"Awards\": {\"N\": \"8\"}}"
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'No One You Know','SongTitle':'Call Me Today',
 'AlbumTitle':'Somewhat Famous', 'Awards':'1'}"

aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'No One You Know','SongTitle':'Howdy',
 'AlbumTitle':'Somewhat Famous', 'Awards':'2'}"

aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'Acme Band','SongTitle':'Happy Day', 'AlbumTitle':'Songs
 About Life', 'Awards':'10'}"

aws dynamodb execute-statement --statement "INSERT INTO Music  \
                VALUE  \
                {'Artist':'Acme Band','SongTitle':'PartiQL Rocks',
 'AlbumTitle':'Another Album Title', 'Awards':'8'}"
```

CMD de Windows

```
aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'No
 One You Know','SongTitle':'Call Me Today', 'AlbumTitle':'Somewhat Famous',
 'Awards':'1'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'No
 One You Know','SongTitle':'Howdy', 'AlbumTitle':'Somewhat Famous', 'Awards':'2'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'Acme
 Band','SongTitle':'Happy Day', 'AlbumTitle':'Songs About Life', 'Awards':'10'}"

aws dynamodb execute-statement --statement "INSERT INTO Music VALUE {'Artist':'Acme
 Band','SongTitle':'PartiQL Rocks', 'AlbumTitle':'Another Album Title',
 'Awards':'8'}"
```

Para obtener más información sobre la escritura de datos con PartiQL, consulte Instrucciones insert de PartiQL.

Para obtener más información sobre los tipos de datos admitidos en DynamoDB, consulte Tipos de datos.

Para obtener más información sobre cómo representar los tipos de datos de DynamoDB en JSON, consulte Valores de los atributos.

AWS SDK

En los siguientes ejemplos de código se muestra cómo escribir un elemento en una tabla de DynamoDB con un SDK de AWS.

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
```

```
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }

  while getopts "n:i:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
```

```
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""
  iecho ""

  response=$(aws dynamodb put-item \
    --table-name "$table_name" \
    --item file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports put-item operation failed.$response"
    return 1
  fi

  return 0

}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
```

```
# the global variable $VERBOSE is set to true.
########################################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

########################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
########################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

########################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
########################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
```

```
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información sobre la API, consulte PutItem en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Put an item in an Amazon DynamoDB table.
/*!
  \sa putItem()
  \param tableName: The table name.
  \param artistKey: The artist key. This is the partition key for the table.
  \param artistValue: The artist value.
  \param albumTitleKey: The album title key.
  \param albumTitleValue: The album title value.
  \param awardsKey: The awards key.
  \param awardsValue: The awards value.
  \param songTitleKey: The song title key.
  \param songTitleValue: The song title value.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::putItem(const Aws::String &tableName,
                               const Aws::String &artistKey,
```

```
                                const Aws::String &artistValue,
                                const Aws::String &albumTitleKey,
                                const Aws::String &albumTitleValue,
                                const Aws::String &awardsKey,
                                const Aws::String &awardsValue,
                                const Aws::String &songTitleKey,
                                const Aws::String &songTitleValue,
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::PutItemRequest putItemRequest;
    putItemRequest.SetTableName(tableName);

    putItemRequest.AddItem(artistKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            artistValue)); // This is the hash key.
    putItemRequest.AddItem(albumTitleKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            albumTitleValue));
    putItemRequest.AddItem(awardsKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(awardsValue));
    putItemRequest.AddItem(songTitleKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(songTitleValue));

    const Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
            putItemRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully added Item!" << std::endl;
    }
    else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Código que espera a que se active la tabla.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Ejemplo 1: Adición de un elemento a una tabla

En el siguiente ejemplo `put-item`, se añade un elemento nuevo a la tabla MusicCollection.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Contenido de `item.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Greatest Hits"}
}
```

Salida:

```
{
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
```

```
        ]
    }
}
```

Para obtener más información, consulte Escritura de un elemento en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 2: Sobrescritura condicional de un elemento de una tabla

En el siguiente ejemplo de `put-item` se sobrescribe un elemento existente de la tabla `MusicCollection` solo si ese elemento existente tiene un atributo `AlbumTitle` con un valor de `Greatest Hits`. El comando devuelve el valor anterior del elemento.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --condition-expression "#A = :A" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

Contenido de `item.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Somewhat Famous"}
}
```

Contenido de `names.json`:

```
{
    "#A": "AlbumTitle"
}
```

Contenido de `values.json`:

```
{
    ":A": {"S": "Greatest Hits"}
}
```

Salida:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Greatest Hits"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Call Me Today"
        }
    }
}
```

Si la clave ya existe, debería ver el siguiente resultado:

```
A client error (ConditionalCheckFailedException) occurred when calling the
 PutItem operation: The conditional request failed.
```

Para obtener más información, consulte Escritura de un elemento en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte PutItem en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
```

```
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}
```

Defina una estructura Película para utilizar en este ejemplo.

```
import (
```

```
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Coloca un elemento en una tabla mediante DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
public class PutItem {
    public static void main(String[] args) {
        final String usage = """
```

```
                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval>
<awards> <awardsval> <Songtitle> <songtitleval>

                Where:
                    tableName - The Amazon DynamoDB table in which an item is
placed (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
Artist).
                    keyval - The key value that represents the item to get (for
example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs
About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
                    SongTitle - The song title (for example, SongTitle).
                    SongTitleVal - The value of the song title (for example,
Happy Day).
                **Warning** This program will  place an item that you specify
into a table!
                """;

        if (args.length != 9) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String albumTitle = args[3];
        String albumTitleValue = args[4];
        String awards = args[5];
        String awardVal = args[6];
        String songTitle = args[7];
        String songTitleVal = args[8];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
```

```
        putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue,
awards, awardVal, songTitle,
                songTitleVal);
        System.out.println("Done!");
        ddb.close();
    }

    public static void putItemInTable(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String albumTitle,
            String albumTitleValue,
            String awards,
            String awardVal,
            String songTitle,
            String songTitleVal) {

        HashMap<String, AttributeValue> itemValues = new HashMap<>();
        itemValues.put(key, AttributeValue.builder().s(keyVal).build());
        itemValues.put(songTitle,
AttributeValue.builder().s(songTitleVal).build());
        itemValues.put(albumTitle,
AttributeValue.builder().s(albumTitleValue).build());
        itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();

        try {
            PutItemResponse response = ddb.putItem(request);
            System.out.println(tableName + " was successfully updated. The
request id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
name correctly!");
            System.exit(1);
        } catch (DynamoDbException e) {
```

```
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener información sobre la API, consulte PutCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Colocar un elemento en una tabla.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};

// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Colocar un elemento en una tabla con el cliente de documentos de DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun putItemInTable(
    tableNameVal: String,
    key: String,
    keyVal: String,
    albumTitle: String,
    albumTitleValue: String,
    awards: String,
    awardVal: String,
    songTitle: String,
    songTitleVal: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()

    // Add all content to the table.
    itemValues[key] = AttributeValue.S(keyVal)
    itemValues[songTitle] = AttributeValue.S(songTitleVal)
    itemValues[albumTitle] = AttributeValue.S(albumTitleValue)
    itemValues[awards] = AttributeValue.S(awardVal)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println(" A new item was placed into $tableNameVal.")
    }
}
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

    public function putItem(array $array)
    {
        $this->dynamoDbClient->putItem($array);
    }
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para PHP.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: crea un nuevo elemento o sustituye un elemento existente por uno nuevo.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- Para obtener detalles de la API, consulte PutItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: crea un nuevo elemento o sustituye un elemento existente por uno nuevo.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- Para obtener detalles de la API, consulte PutItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def add_movie(self, title, year, plot, rating):
        """
        Adds a movie to the table.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :param plot: The plot summary of the movie.
        :param rating: The quality rating of the movie.
        """
        try:
            self.table.put_item(
                Item={
                    "year": year,
                    "title": title,
                    "info": {"plot": plot, "rating": Decimal(str(rating))},
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't add movie %s to table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Adds a movie to the table.
  #
  # @param movie [Hash] The title, year, plot, and rating of the movie.
  def add_item(movie)
    @table.put_item(
      item: {
        'year' => movie[:year],
        'title' => movie[:title],
        'info' => { 'plot' => movie[:plot], 'rating' => movie[:rating] }
      }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't add movie #{title} to table #{@table.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
pub async fn add_item(client: &Client, item: Item, table: &String) ->
 Result<ItemOut, Error> {
    let user_av = AttributeValue::S(item.username);
    let type_av = AttributeValue::S(item.p_type);
    let age_av = AttributeValue::S(item.age);
    let first_av = AttributeValue::S(item.first);
    let last_av = AttributeValue::S(item.last);

    let request = client
        .put_item()
        .table_name(table)
        .item("username", user_av)
        .item("account_type", type_av)
        .item("age", age_av)
        .item("first_name", first_av)
        .item("last_name", last_av);

    println!("Executing request [{request:?}] to add item...");

    let resp = request.send().await?;

    let attributes = resp.attributes().unwrap();

    let username = attributes.get("username").cloned();
    let first_name = attributes.get("first_name").cloned();
    let last_name = attributes.get("last_name").cloned();
    let age = attributes.get("age").cloned();
    let p_type = attributes.get("p_type").cloned();

    println!(
        "Added user {:?}, {:?} {:?}, age {:?} as {:?} user",
        username, first_name, last_name, age, p_type
    );

    Ok(ItemOut {
        p_type,
        age,
        username,
        first_name,
        last_name,
    })
}
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    TRY.
        DATA(lo_resp) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = it_item ).
        MESSAGE '1 row inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB


    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    ///
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Get a DynamoDB item containing the movie data.
            let item = try await movie.getAsItem()

            // Send the `PutItem` request to Amazon DynamoDB.

            let input = PutItemInput(
                item: item,
                tableName: self.tableName
            )
            _ = try await client.putItem(input: input)
        } catch {
            print("ERROR: add movie:", dump(error))
            throw error
        }
    }


    ///
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
 [Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.
```

```
        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
```

- Para obtener información acerca de la API, consulte PutItem en la Referencia de la API del SDK de AWS para Swift.

Para obtener más ejemplos de DynamoDB, consulte Ejemplos de código de DynamoDB con los SDK de AWS.

Después de escribir los datos en la tabla, continúe en el Paso 3: leer datos de una tabla de DynamoDB.

Paso 3: leer datos de una tabla de DynamoDB

En este paso, volverá a leer uno de los elementos que creó en Paso 2: escribir datos en una tabla de DynamoDB. Puede utilizar la consola de DynamoDB o la AWS CLI para leer un elemento de la tabla `Music` especificando los valores de los campos `Artist` y `SongTitle`.

Para obtener más información sobre las operaciones de lectura en DynamoDB, consulte Lectura de un elemento.

Consola de administración de AWS

Siga estos pasos para leer datos de la tabla `Music` mediante la consola de DynamoDB.

1. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación izquierdo, elija Tables (Tablas).
3. En la página Tablas, elija la tabla Música.
4. Elija Explorar elementos de la tabla.
5. En la sección Elementos devueltos, puede ver la lista de los elementos almacenados en la tabla, ordenados por `Artist` y `SongTitle`. El primer elemento de la lista es Artista llamado Acme Band y SongTitle PartiQL Rocks.

AWS CLI

En el siguiente ejemplo de AWS CLI, se lee un elemento de la tabla mediante `Music`. Puede hacerlo mediante la API de DynamoDB o PartiQL, un lenguaje de consulta compatible con SQL para DynamoDB.

DynamoDB API

> **Note**
>
> El comportamiento predeterminado de DynamoDB es el de lectura eventualmente consistentes. El parámetro `consistent-read` se utiliza a continuación para demostrar las lecturas de consistencia alta.

Linux

```
aws dynamodb get-item --consistent-read \
    --table-name Music \
    --key '{ "Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"}}'
```

CMD de Windows

```
aws dynamodb get-item --consistent-read ^
    --table-name Music ^
```

```
    --key "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day
\"}}"
```

El uso de `get-item` devuelve el siguiente resultado de ejemplo.

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "Awards": {
            "S": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    }
}
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
WHERE Artist='Acme Band' AND SongTitle='Happy Day'"
```

CMD de Windows

```
aws dynamodb execute-statement --statement "SELECT * FROM Music WHERE Artist='Acme
 Band' AND SongTitle='Happy Day'"
```

El uso de la instrucción `Select` de PartiQL devuelve el siguiente resultado de ejemplo.

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Songs About Life"
            },
```

```
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        }
    ]
}
```

Para obtener más información sobre la lectura de datos con PartiQL, consulte Instrucciones select de PartiQL.

AWS SDK

En los siguientes ejemplos de código se muestra cómo leer un elemento de una tabla de DynamoDB con un SDK de AWS.

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</
param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
```

```
   public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
newMovie, string tableName)
   {
       try
       {
           var key = new Dictionary<string, AttributeValue>
           {
               ["title"] = new AttributeValue { S = newMovie.Title },
               ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
           };

           var request = new GetItemRequest
           {
               Key = key,
               TableName = tableName,
           };

           var response = await _amazonDynamoDB.GetItemAsync(request);
           return response.Item;
       }
       catch (ResourceNotFoundException ex)
       {
           Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
           return new Dictionary<string, AttributeValue>();
       }
       catch (AmazonDynamoDBException ex)
       {
           Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
           throw;
       }
       catch (Exception ex)
       {
           Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
           throw;
       }
   }
```

- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_get_item"
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to get."
    echo " [-q query]  -- Optional JMESPath query expression."
    echo ""
```

```
}
query=""
while getopts "n:k:q:h" option; do
  case "${option}" in
    n) table_name="${OPTARG}" ;;
    k) keys="${OPTARG}" ;;
    q) query="${OPTARG}" ;;
    h)
      usage
      return 0
      ;;
    \?)
      echo "Invalid parameter"
      usage
      return 1
      ;;
  esac
done
export OPTIND=1

if [[ -z "$table_name" ]]; then
  errecho "ERROR: You must provide a table name with the -n parameter."
  usage
  return 1
fi

if [[ -z "$keys" ]]; then
  errecho "ERROR: You must provide a keys json file path the -k parameter."
  usage
  return 1
fi

if [[ -n "$query" ]]; then
  response=$(aws dynamodb get-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --output text \
    --query "$query")
else
  response=$(
    aws dynamodb get-item \
      --table-name "$table_name" \
      --key file://"$keys" \
      --output text
```

```
    )
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
  fi

  if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
```

```
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información sobre la API, consulte GetItem en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Get an item from an Amazon DynamoDB table.
/*!
  \sa getItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::getItem(const Aws::String &tableName,
                               const Aws::String &partitionKey,
                               const Aws::String &partitionValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::GetItemRequest request;

    // Set up the request.
    request.SetTableName(tableName);
    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));

    // Retrieve the item's fields and values.
    const Aws::DynamoDB::Model::GetItemOutcome &outcome =
 dynamoClient.GetItem(request);
    if (outcome.IsSuccess()) {
        // Reference the retrieved fields/values.
        const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> &item =
 outcome.GetResult().GetItem();
        if (!item.empty()) {
            // Output each retrieved field and its value.
            for (const auto &i: item)
                std::cout << "Values: " << i.first << ": " << i.second.GetS()
                          << std::endl;
        }
        else {
            std::cout << "No item found with the key " << partitionKey <<
 std::endl;
        }
    }
    else {
        std::cerr << "Failed to get item: " << outcome.GetError().GetMessage();
```

```
    }

    return outcome.IsSuccess();
}
```

- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Ejemplo 1: Lectura de un elemento de una tabla

En el siguiente ejemplo `get-item`, se recupera un elemento de la tabla `MusicCollection`. La tabla tiene una clave principal hash y de rango (`Artist` y `SongTitle`), por lo que debe especificar ambos atributos. El comando también solicita información sobre la capacidad de lectura consumida por la operación.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-consumed-capacity TOTAL
```

Contenido de `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Salida:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
```

```
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Para obtener más información, consulte Lectura de un elemento en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 2: Lectura de un elemento mediante una lectura coherente

En el siguiente ejemplo, se recupera un elemento de la tabla `MusicCollection` con lecturas altamente coherentes.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --consistent-read \
    --return-consumed-capacity TOTAL
```

Contenido de `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Salida:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
```

```
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Para obtener más información, consulte Lectura de un elemento en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 3: recuperación de atributos específicos de un elemento

En el siguiente ejemplo, se utiliza una expresión de proyección para recuperar solo tres atributos del elemento deseado.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "102"}}' \
    --projection-expression "#T, #C, #P" \
    --expression-attribute-names file://names.json
```

Contenido de `names.json`:

```
{
    "#T": "Title",
    "#C": "ProductCategory",
    "#P": "Price"
}
```

Salida:

```
{
    "Item": {
        "Price": {
            "N": "20"
        },
        "Title": {
            "S": "Book 102 Title"
        },
```

```
        "ProductCategory": {
            "S": "Book"
        }
    }
}
```

Para obtener más información, consulte Lectura de un elemento en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte GetItem en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
```

```
 TableName      string
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

Defina una estructura Película para utilizar en este ejemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)
```

```
// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Obtiene un elemento de una tabla mediante DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal>

                Where:
```

```
                    tableName - The Amazon DynamoDB table from which an item is
retrieved (for example, Music3).\s
                    key - The key used in the Amazon DynamoDB table (for example,
Artist).\s
                    keyval - The key value that represents the item to get (for
example, Famous Band).
                """";

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal,
tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        getDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void getDynamoDBItem(DynamoDbClient ddb, String tableName,
String key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        GetItemRequest request = GetItemRequest.builder()
                .key(keyToGet)
                .tableName(tableName)
                .build();

        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem =
ddb.getItem(request).item();
            if (returnedItem.isEmpty())
```

```
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener información sobre la API, consulte GetCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
```

```
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Obtener un elemento de una tabla.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
```

```
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

Obtener un elemento de una tabla con el cliente de documentos de DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun getSpecificItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

- Para obtener información de la API, consulte GetItem en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
{$movie['Item']['year']['N']}.\n";

    public function getItemByKey(string $tableName, array $key)
    {
        return $this->dynamoDbClient->getItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK para PHP.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: devuelve el elemento de DynamoDB con la clave de partición SongTitle y la clave de clasificación Artist.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Salida:

```
Name                            Value
----                            -----
Genre                           Country
SongTitle                       Somewhere Down The Road
Price                           1.94
Artist                          No One You Know
CriticRating                    9
AlbumTitle                      Somewhat Famous
```

- Para obtener detalles de la API, consulte GetItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: devuelve el elemento de DynamoDB con la clave de partición SongTitle y la clave de clasificación Artist.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Salida:

```
Name                            Value
----                            -----
Genre                           Country
SongTitle                       Somewhere Down The Road
Price                           1.94
Artist                          No One You Know
CriticRating                    9
AlbumTitle                      Somewhat Famous
```

- Para obtener detalles de la API, consulte GetItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]
```

- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table
```

```
def initialize(table_name)
  client = Aws::DynamoDB::Client.new(region: 'us-east-1')
  @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
  @table = @dynamo_resource.table(table_name)
end

# Gets movie data from the table for a specific movie.
#
# @param title [String] The title of the movie.
# @param year [Integer] The release year of the movie.
# @return [Hash] The data about the requested movie.
def get_item(title, year)
  @table.get_item(key: { 'year' => year, 'title' => title })
rescue Aws::DynamoDB::Errors::ServiceError => e
  puts("Couldn't get movie #{title} (#{year}) from table #{@table.name}:\n")
  puts("\t#{e.code}: #{e.message}")
  raise
end
```

- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK para Ruby.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
TRY.
    oo_item = lo_dyn->getitem(
      iv_tablename               = iv_table_name
      it_key                     = it_key ).
    DATA(lt_attr) = oo_item->get_item( ).
    DATA(lo_title) = lt_attr[ key = 'title' ]-value.
    DATA(lo_year) = lt_attr[ key = 'year' ]-value.
    DATA(lo_rating) = lt_attr[ key = 'rating' ]-value.
```

```
        MESSAGE 'Movie name is: ' && lo_title->get_s( )
          && 'Movie year is: ' && lo_year->get_n( )
          && 'Moving rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Para obtener información sobre la API, consulte GetItem en la Referencia de la API de AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB


    /// Return a `Movie` record describing the specified movie from the Amazon
    /// DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The movie's release year (`Int`).
    ///
    /// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
    ///
    /// - Returns: A `Movie` record with the movie's details.
    func get(title: String, year: Int) async throws -> Movie {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = GetItemInput(
```

```
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            let output = try await client.getItem(input: input)
            guard let item = output.item else {
                throw MoviesError.ItemNotFound
            }

            let movie = try Movie(withItem: item)
            return movie
        } catch {
            print("ERROR: get:", dump(error))
            throw error
        }
    }
```

- Para obtener detalles sobre la API, consulte GetItem en la Referencia de la API del SDK de AWS para Swift.

Para obtener más ejemplos de DynamoDB, consulte Ejemplos de código de DynamoDB con los SDK de AWS.

Para actualizar los datos de la tabla, continúe en el Paso 4: actualizar los datos de una tabla de DynamoDB.

Paso 4: actualizar los datos de una tabla de DynamoDB

En este paso, actualizará un elemento que creó en el Paso 2: escribir datos en una tabla de DynamoDB. Puede utilizar la consola de DynamoDB o la AWS CLI para actualizar el campo `AlbumTitle` de un elemento de la tabla `Music` especificando los valores de los campos `Artist` y `SongTitle`, y el valor actualizado del campo `AlbumTitle`.

Para obtener más información sobre las operaciones de escritura, consulte Escritura de un elemento.

Consola de administración de AWS

Puede utilizar la consola de DynamoDB para actualizar los datos de la tabla `Music`.

1. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación izquierdo, elija Tables (Tablas).
3. Elija la tabla Music (Música) en la lista de tablas.
4. Elija Explorar elementos de la tabla.
5. En Elementos devueltos, para la fila de elementos con Acme Band Artista y Happy Day SongTitle, haga lo siguiente:

   a. Coloque el cursor sobre el AlbumTitle llamado Songs About Life.
   b. Elija el icono de edición.
   c. En la ventana emergente Edición de cadena, ingrese **`Songs of Twilight`**.
   d. Seleccione Save.

> **Tip**
>
> Como alternativa, para actualizar un elemento, haga lo siguiente en la sección Elementos devueltos:
>
> 1. Elija la fila de elementos con Artista llamado Acme Band y SongTitle llamado Happy Day.
> 2. En la lista desplegable de Acciones, elija Edición de elemento.
> 3. Para ingresar AlbumTitle, ingrese **`Songs of Twilight`**.
> 4. Elija Save and close.

AWS CLI

En el siguiente ejemplo de AWS CLI, se actualiza un elemento de la tabla `Music`. Puede hacerlo mediante la API de DynamoDB o PartiQL, un lenguaje de consulta compatible con SQL para DynamoDB.

DynamoDB API

Linux

```
aws dynamodb update-item \
    --table-name Music \
    --key '{ "Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"}}' \
    --update-expression "SET AlbumTitle = :newval" \
    --expression-attribute-values '{":newval":{"S":"Updated Album Title"}}' \
    --return-values ALL_NEW
```

CMD de Windows

```
aws dynamodb update-item ^
    --table-name Music ^
    --key "{\"Artist\": {\"S\": \"Acme Band\"}, \"SongTitle\": {\"S\": \"Happy Day
\"}}" ^
    --update-expression "SET AlbumTitle = :newval" ^
    --expression-attribute-values "{\":newval\":{\"S\":\"Updated Album Title\"}}" ^
    --return-values ALL_NEW
```

El uso de `update-item` devuelve el siguiente resultado de ejemplo porque `return-values ALL_NEW` se especificó.

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Updated Album Title"
        },
        "Awards": {
            "S": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    }
}
```

PartiQL for DynamoDB

Linux

```
aws dynamodb execute-statement --statement "UPDATE Music  \
```

```
    SET AlbumTitle='Updated Album Title'  \
    WHERE Artist='Acme Band' AND SongTitle='Happy Day' \
    RETURNING ALL NEW *"
```

CMD de Windows

```
aws dynamodb execute-statement --statement "UPDATE Music SET AlbumTitle='Updated
 Album Title' WHERE Artist='Acme Band' AND SongTitle='Happy Day' RETURNING ALL NEW
 *"
```

El uso de la instrucción `Update` devuelve el siguiente resultado de ejemplo porque `RETURNING ALL NEW *` se especificó.

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        }
    ]
}
```

Para obtener más información sobre la actualización de datos con PartiQL, consulte Instrucciones update de PartiQL.

AWS SDK

En los siguientes ejemplos de código se muestra cómo actualizar un elemento de una tabla de DynamoDB con un SDK de AWS.

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },
```

```
                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_update_item"
    echo "Update an item in a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to update."
```

```
    echo " -e update expression  -- An expression that defines one or more
attributes to be updated."
    echo " -v values  -- Path to json file containing the update values."
    echo ""
 }

 while getopts "n:k:e:v:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      e) update_expression="${OPTARG}" ;;
      v) values="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
 fi

 if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
 fi
 if [[ -z "$update_expression" ]]; then
    errecho "ERROR: You must provide an update expression with the -e parameter."
    usage
    return 1
 fi

 if [[ -z "$values" ]]; then
    errecho "ERROR: You must provide a values json file path the -v parameter."
```

```
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho "    update_expression:   $update_expression"
  iecho "    values:   $values"

  response=$(aws dynamodb update-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --update-expression "$update_expression" \
    --expression-attribute-values file://"$values")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
  fi

  return 0

}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}
```

```
#############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
#############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
```

```
}
```

- Para obtener información sobre la API, consulte UpdateItem en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Update an Amazon DynamoDB table item.
/*!
  \sa updateItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param attributeKey: The key for the attribute to be updated.
  \param attributeValue: The value for the attribute to be updated.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 *  The example code only sets/updates an attribute value. It processes
 *  the attribute value as a string, even if the value could be interpreted
 *  as a number. Also, the example code does not remove an existing attribute
 *  from the key value.
 */

bool AwsDoc::DynamoDB::updateItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &attributeKey,
                                  const Aws::String &attributeValue,
```

```
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // *** Define UpdateItem request arguments.
    // Define TableName argument.
    Aws::DynamoDB::Model::UpdateItemRequest request;
    request.SetTableName(tableName);

    // Define KeyName argument.
    Aws::DynamoDB::Model::AttributeValue attribValue;
    attribValue.SetS(partitionValue);
    request.AddKey(partitionKey, attribValue);

    // Construct the SET update expression argument.
    Aws::String update_expression("SET #a = :valueA");
    request.SetUpdateExpression(update_expression);

    // Construct attribute name argument.
    Aws::Map<Aws::String, Aws::String> expressionAttributeNames;
    expressionAttributeNames["#a"] = attributeKey;
    request.SetExpressionAttributeNames(expressionAttributeNames);

    // Construct attribute value argument.
    Aws::DynamoDB::Model::AttributeValue attributeUpdatedValue;
    attributeUpdatedValue.SetS(attributeValue);
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 expressionAttributeValues;
    expressionAttributeValues[":valueA"] = attributeUpdatedValue;
    request.SetExpressionAttributeValues(expressionAttributeValues);

    // Update the item.
    const Aws::DynamoDB::Model::UpdateItemOutcome &outcome =
 dynamoClient.UpdateItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item was updated" << std::endl;
    } else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Código que espera a que se active la tabla.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
```

```
    return false;
}
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Ejemplo 1: Actualización de un elemento de una tabla

En el siguiente ejemplo de `update-item`, se actualiza un elemento de la tabla `MusicCollection`. Añade un nuevo atributo (`Year`) y modifica el atributo `AlbumTitle`. En la respuesta se muestran todos los atributos del elemento, tal como aparecen después de la actualización.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --return-values ALL_NEW \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Contenido de `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Contenido de `expression-attribute-names.json`:

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
}
```

Contenido de `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Salida:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Louder Than Ever"
        },
        "Awards": {
            "N": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "Year": {
            "N": "2015"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 3.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "Acme Band"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Para obtener más información, consulte Escritura de un elemento en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 2: actualización de un elemento de forma condicional

En el siguiente ejemplo se actualiza un elemento de la tabla `MusicCollection`, pero solo si el elemento existente aún no tiene un atributo `Year`.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --condition-expression "attribute_not_exists(#Y)"
```

Contenido de `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Contenido de `expression-attribute-names.json`:

```
{
    "#Y":"Year",
    "#AT":"AlbumTitle"
}
```

Contenido de `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Si el elemento ya tiene un atributo `Year`, DynamoDB devuelve el siguiente resultado.

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
 operation: The conditional request failed
```

Para obtener más información, consulte Escritura de un elemento en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte UpdateItem en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
```

```
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx,
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}
```

Defina una estructura Película para utilizar en este ejemplo.

```
import (
 "archive/zip"
```

```
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Actualiza un elemento de una tabla mediante DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To update an Amazon DynamoDB table using the AWS SDK for Java V2, its better
 * practice to use the
 * Enhanced Client, See the EnhancedModifyItem example.
 */
public class UpdateItem {
    public static void main(String[] args) {
        final String usage = """
```

```
                Usage:
                    <tableName> <key> <keyVal> <name> <updateVal>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music3).
                    key - The name of the key in the table (for example, Artist).
                    keyVal - The value of the key (for example, Famous Band).
                    name - The name of the column where the value is updated (for
example, Awards).
                    updateVal - The value used to update an item (for example,
14).
                 Example:
                    UpdateItem Music3 Artist Famous Band Awards 14
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String name = args[3];
        String updateVal = args[4];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        updateTableItem(ddb, tableName, key, keyVal, name, updateVal);
        ddb.close();
    }

    public static void updateTableItem(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String name,
            String updateVal) {

        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put(key, AttributeValue.builder()
```

```
                .s(keyVal)
                .build());

        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put(name, AttributeValueUpdate.builder()
                .value(AttributeValue.builder().s(updateVal).build())
                .action(AttributeAction.PUT)
                .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(itemKey)
                .attributeUpdates(updatedValues)
                .build();

        try {
            ddb.updateItem(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("The Amazon DynamoDB table was updated!");
    }
}
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener información sobre la API, consulte UpdateCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun updateTableItem(
    tableNameVal: String,
    keyName: String,
```

```
    keyVal: String,
    name: String,
    updateVal: String,
) {
    val itemKey = mutableMapOf<String, AttributeValue>()
    itemKey[keyName] = AttributeValue.S(keyVal)

    val updatedValues = mutableMapOf<String, AttributeValueUpdate>()
    updatedValues[name] =
        AttributeValueUpdate {
            value = AttributeValue.S(updateVal)
            action = AttributeAction.Put
        }

    val request =
        UpdateItemRequest {
            tableName = tableNameVal
            key = itemKey
            attributeUpdates = updatedValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.updateItem(request)
        println("Item in $tableNameVal was updated")
    }
}
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

    public function updateItemAttributeByKey(
        string $tableName,
        array $key,
        string $attributeName,
        string $attributeType,
        string $newValue
    ) {
        $this->dynamoDbClient->updateItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
            'UpdateExpression' => "set #NV=:NV",
            'ExpressionAttributeNames' => [
                '#NV' => $attributeName,
            ],
            'ExpressionAttributeValues' => [
                ':NV' => [
                    $attributeType => $newValue
                ]
            ],
        ]);
    }
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para PHP.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: establece el atributo de género como Rap en el elemento de DynamoDB con la clave de partición SongTitle y la clave de clasificación Artist.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

Salida:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Para obtener detalles de la API, consulte UpdateItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: establece el atributo de género como Rap en el elemento de DynamoDB con la clave de partición SongTitle y la clave de clasificación Artist.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
```

```
Update-DDBItem @updateDdbItem
```

Salida:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Para obtener detalles de la API, consulte UpdateItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Actualizar un elemento con una expresión de actualización.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
```

```
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

Actualizar un elemento con una expresión de actualización que incluye una operación aritmética.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def update_rating(self, title, year, rating_change):
        """
        Updates the quality rating of a movie in the table by using an arithmetic
        operation in the update expression. By specifying an arithmetic
 operation,
        you can adjust a value in a single request, rather than first getting its
        value and then setting its new value.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating_change: The amount to add to the current rating for the
 movie.
        :return: The updated rating.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating = info.rating + :val",
                ExpressionAttributeValues={":val": Decimal(str(rating_change))},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

Actualizar un elemento solo cuando cumpla determinadas condiciones.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def remove_actors(self, title, year, actor_threshold):
        """
        Removes an actor from a movie, but only when the number of actors is
 greater
        than a specified threshold. If the movie does not list more than the
 threshold,
        no actors are removed.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param actor_threshold: The threshold of actors to check.
        :return: The movie data after the update.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="remove info.actors[0]",
                ConditionExpression="size(info.actors) > :num",
                ExpressionAttributeValues={":num": actor_threshold},
                ReturnValues="ALL_NEW",
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "ConditionalCheckFailedException":
                logger.warning(
                    "Didn't update %s because it has fewer than %s actors.",
                    title,
                    actor_threshold + 1,
                )
            else:
                logger.error(
                    "Couldn't update movie %s. Here's why: %s: %s",
                    title,
```

```
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response["Attributes"]
```

- Para obtener información la API, consulte UpdateItem en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Updates rating and plot data for a movie in the table.
  #
  # @param movie [Hash] The title, year, plot, rating of the movie.
  def update_item(movie)
    response = @table.update_item(
      key: { 'year' => movie[:year], 'title' => movie[:title] },
      update_expression: 'set info.rating=:r',
      expression_attribute_values: { ':r' => movie[:rating] },
      return_values: 'UPDATED_NEW'
    )
```

```
 rescue Aws::DynamoDB::Errors::ServiceError => e
   puts("Couldn't update movie #{movie[:title]} (#{movie[:year]}) in table
#{@table.name}\n")
   puts("\t#{e.code}: #{e.message}")
   raise
 else
   response.attributes
 end
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para Ruby.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
   TRY.
       oo_output = lo_dyn->updateitem(
         iv_tablename        = iv_table_name
         it_key              = it_item_key
         it_attributeupdates = it_attribute_updates ).
       MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
     CATCH /aws1/cx_dyncondalcheckfaile00.
       MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
     CATCH /aws1/cx_dynresourcenotfoundex.
       MESSAGE 'The table or index does not exist' TYPE 'E'.
     CATCH /aws1/cx_dyntransactconflictex.
       MESSAGE 'Another transaction is using the item' TYPE 'E'.
   ENDTRY.
```

- Para obtener información sobre la API, consulte UpdateItem en la Referencia de la API del AWSSDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB


    /// Update the specified movie with new `rating` and `plot` information.
    ///
    /// - Parameters:
    ///   - title: The title of the movie to update.
    ///   - year: The release year of the movie to update.
    ///   - rating: The new rating for the movie.
    ///   - plot: The new plot summary string for the movie.
    ///
    /// - Returns: An array of mappings of attribute names to their new
    ///   listing each item actually changed. Items that didn't need to change
    ///   aren't included in this list. `nil` if no changes were made.
    ///
    func update(title: String, year: Int, rating: Double? = nil, plot: String? =
 nil) async throws
        -> [Swift.String: DynamoDBClientTypes.AttributeValue]?
    {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Build the update expression and the list of expression attribute
            // values. Include only the information that's changed.

            var expressionParts: [String] = []
            var attrValues: [Swift.String: DynamoDBClientTypes.AttributeValue] =
 [:]

            if rating != nil {
```

```
                expressionParts.append("info.rating=:r")
                attrValues[":r"] = .n(String(rating!))
            }
            if plot != nil {
                expressionParts.append("info.plot=:p")
                attrValues[":p"] = .s(plot!)
            }
            let expression = "set \(expressionParts.joined(separator: ", "))"

            let input = UpdateItemInput(
                // Create substitution tokens for the attribute values, to ensure
                // no conflicts in expression syntax.
                expressionAttributeValues: attrValues,
                // The key identifying the movie to update consists of the
release
                // year and title.
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                returnValues: .updatedNew,
                tableName: self.tableName,
                updateExpression: expression
            )
            let output = try await client.updateItem(input: input)

            guard let attributes: [Swift.String:
DynamoDBClientTypes.AttributeValue] = output.attributes else {
                throw MoviesError.InvalidAttributes
            }
            return attributes
        } catch {
            print("ERROR: update:", dump(error))
            throw error
        }
    }
```

- Para obtener detalles de la API, consulte UpdateItem en la referencia de la API del SDK de AWS para Swift.

Para obtener más ejemplos de DynamoDB, consulte Ejemplos de código de DynamoDB con los SDK de AWS.

Para consultar los datos de la tabla `Music`, continúe en el Paso 5: consultar los datos de una tabla de DynamoDB.

Paso 5: consultar los datos de una tabla de DynamoDB

En este paso, consultará los datos que escribió en la tabla `Music` en el the section called “Paso 2: escribir datos” especificando el campo `Artist`. Se mostrarán todas las canciones asociadas a la clave de partición .: `Artist`.

Para obtener más información sobre las operaciones de consulta, consulte Consulta de tablas en DynamoDB.

Consola de administración de AWS

Siga estos pasos para utilizar la consola de DynamoDB para consultar los datos de la tabla `Music`.

1. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación izquierdo, elija Tables (Tablas).
3. Elija la tabla Music (Música) en la lista de tablas.
4. Elija Explorar elementos de la tabla.
5. En Escaneo o consulta de elementos, asegúrese de que Consulta esté seleccionado.
6. En Partition key (Clave de partición), ingrese **`Acme Band`** y, a continuación, elija Run (Ejecutar).

AWS CLI

En el siguiente ejemplo de AWS CLI se consulta un elemento de la tabla `Music`. Puede hacerlo mediante la API de DynamoDB o PartiQL, un lenguaje de consulta compatible con SQL para DynamoDB.

DynamoDB API

> El usuario consulta un elemento a través de la API de DynamoDB mediante `query` y al proporcionar la clave de partición.
>
> Linux

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression "Artist = :name" \
    --expression-attribute-values  '{":name":{"S":"Acme Band"}}'
```

CMD de Windows

```
aws dynamodb query ^
    --table-name Music ^
    --key-condition-expression "Artist = :name" ^
    --expression-attribute-values  "{\":name\":{\"S\":\"Acme Band\"}}"
```

El uso de `query` devuelve todas las canciones asociadas a este `Artist` en particular.

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "N": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        },
        {
            "AlbumTitle": {
                "S": "Another Album Title"
            },
            "Awards": {
                "N": "8"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
```

```
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

PartiQL for DynamoDB

El usuario consulta un elemento a través de PartiQL mediante la instrucción `Select` y al proporcionar la clave de partición.

Linux

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
                                            WHERE Artist='Acme Band'"
```

CMD de Windows

```
aws dynamodb execute-statement --statement "SELECT * FROM Music WHERE Artist='Acme
 Band'"
```

El uso de la instrucción `Select` de esta manera devuelve todas las canciones asociadas a este `Artist` en particular.

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Updated Album Title"
            },
            "Awards": {
                "S": "10"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "Happy Day"
            }
        },
```

```
        {
            "AlbumTitle": {
                "S": "Another Album Title"
            },
            "Awards": {
                "S": "8"
            },
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

Para obtener más información sobre la consulta de datos con PartiQL, consulte Instrucciones select de PartiQL.

AWS SDK

En los siguientes ejemplos de código se muestra cómo consultar una tabla de DynamoDB con un SDK de AWS.

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Queries the table for movies released in a particular year and
    /// then displays the information for the movies returned.
    /// </summary>
    /// <param name="tableName">The name of the table to query.</param>
```

```
    /// <param name="year">The release year for which we want to
    /// view movies.</param>
    /// <returns>The number of movies that match the query.</returns>
    public async Task<int> QueryMoviesAsync(string tableName, int year)
    {
        try
        {
            var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
                .AddHashKey("year", DynamoDBEntryType.Numeric)
                .AddRangeKey("title", DynamoDBEntryType.String)
                .Build();

            var filter = new QueryFilter("year", QueryOperator.Equal, year);

            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
```

```
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_query
#
```

```
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_query() {
  local table_name key_condition_expression attribute_names attribute_values
 projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_query"
    echo "Query a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k key_condition_expression -- The key condition expression."
    echo " -a attribute_names -- Path to JSON file containing the attribute
 names."
    echo " -v attribute_values -- Path to JSON file containing the attribute
 values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:k:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) key_condition_expression="${OPTARG}" ;;
      a) attribute_names="${OPTARG}" ;;
      v) attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
```

```
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$key_condition_expression" ]]; then
    errecho "ERROR: You must provide a key condition expression with the -k
parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_names" ]]; then
    errecho "ERROR: You must provide a attribute names with the -a parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_values" ]]; then
    errecho "ERROR: You must provide a attribute values with the -v parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values")
  else
```

```
    response=$(aws dynamodb query \
      --table-name "$table_name" \
      --key-condition-expression "$key_condition_expression" \
      --expression-attribute-names file://"$attribute_names" \
      --expression-attribute-values file://"$attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports query operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
```

```
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información sobre la API, consulte Query en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Perform a query on an Amazon DynamoDB Table and retrieve items.
/*!
  \sa queryItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param projectionExpression: The projections expression, which is ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 * The partition key attribute is searched with the specified value. By default,
 all fields and values
 * contained in the item are returned. If an optional projection expression is
 * specified on the command line, only the specified fields and values are
 * returned.
 */

bool AwsDoc::DynamoDB::queryItems(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &projectionExpression,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::QueryRequest request;

    request.SetTableName(tableName);

    if (!projectionExpression.empty()) {
        request.SetProjectionExpression(projectionExpression);
    }

    // Set query key condition expression.
    request.SetKeyConditionExpression(partitionKey + "= :valueToMatch");

    // Set Expression AttributeValues.
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> attributeValues;
    attributeValues.emplace(":valueToMatch", partitionValue);

    request.SetExpressionAttributeValues(attributeValues);
```

```
   bool result = true;

   // "exclusiveStartKey" is used for pagination.
   Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
exclusiveStartKey;
   do {
       if (!exclusiveStartKey.empty()) {
           request.SetExclusiveStartKey(exclusiveStartKey);
           exclusiveStartKey.clear();
       }
       // Perform Query operation.
       const Aws::DynamoDB::Model::QueryOutcome &outcome =
dynamoClient.Query(request);
       if (outcome.IsSuccess()) {
           // Reference the retrieved items.
           const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
           if (!items.empty()) {
               std::cout << "Number of items retrieved from Query: " <<
items.size()
                         << std::endl;
               // Iterate each item and print.
               for (const auto &item: items) {
                   std::cout
                           <<
"**************************************************"
                           << std::endl;
                   // Output each retrieved field and its value.
                   for (const auto &i: item)
                       std::cout << i.first << ": " << i.second.GetS() <<
std::endl;
               }
           }
           else {
               std::cout << "No item found in table: " << tableName <<
std::endl;
           }

           exclusiveStartKey = outcome.GetResult().GetLastEvaluatedKey();
       }
       else {
           std::cerr << "Failed to Query items: " <<
outcome.GetError().GetMessage();
```

```
            result = false;
            break;
        }
    } while (!exclusiveStartKey.empty());

    return result;
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Ejemplo 1: Consulta de una tabla

En el siguiente ejemplo de `query` se consultan elementos de la tabla `MusicCollection`. La tabla tiene una clave principal hash y de rango (`Artist` y `SongTitle`), pero esta consulta solo especifica el valor de la clave hash. Devuelve los títulos de las canciones del artista llamado “No One You Know”.

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --return-consumed-capacity TOTAL
```

Contenido de `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Salida:

```
{
    "Items": [
```

```
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Para obtener más información, consulte Uso de consultas en DynamoDB en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 2: Consulta de una tabla con lecturas altamente coherentes y recorrer el índice en orden descendente

En el siguiente ejemplo se realiza la misma consulta que en el primer ejemplo, pero se devuelven los resultados en orden inverso y se utilizan lecturas altamente coherentes.

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --consistent-read \
    --no-scan-index-forward \
    --return-consumed-capacity TOTAL
```

Contenido de `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Salida:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Para obtener más información, consulte Uso de consultas en DynamoDB en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 3: filtrado de resultados específicos

En el siguiente ejemplo se consulta `MusicCollection` pero se excluyen los resultados con valores específicos en el atributo `AlbumTitle`. Tenga en cuenta que esto no afecta a `ScannedCount` o `ConsumedCapacity`, ya que el filtro se aplica después de leer los elementos.

```
aws dynamodb query \
    --table-name MusicCollection \
    --key-condition-expression "#n1 = :v1" \
    --filter-expression "NOT (#n2 IN (:v2, :v3))" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-consumed-capacity TOTAL
```

Contenido de `values.json`:

```
{
```

```
    ":v1": {"S": "No One You Know"},
    ":v2": {"S": "Blue Sky Blues"},
    ":v3": {"S": "Greatest Hits"}
}
```

Contenido de `names.json`:

```
{
    "#n1": "Artist",
    "#n2": "AlbumTitle"
}
```

Salida:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 1,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Para obtener más información, consulte Uso de consultas en DynamoDB en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 4: Recuperación de un solo recuento de elementos

En el siguiente ejemplo, se recupera un recuento de los elementos que coinciden con la consulta, pero no recupera ninguno de los elementos en sí.

```
aws dynamodb query \
    --table-name MusicCollection \
    --select COUNT \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

Contenido de `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Salida:

```
{
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

Para obtener más información, consulte Uso de consultas en DynamoDB en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 5: consulta de un índice

El siguiente ejemplo consulta el índice secundario global `AlbumTitleIndex`. La consulta devuelve todos los atributos de la tabla base que se han proyectado en el índice secundario local. Tenga en cuenta que, al consultar un índice secundario local o un índice secundario global, también debe proporcionar el nombre de la tabla base mediante el parámetro `table-name`.

```
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --select ALL_PROJECTED_ATTRIBUTES \
    --return-consumed-capacity INDEXES
```

Contenido de `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Salida:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5,
        "Table": {
            "CapacityUnits": 0.0
        },
        "LocalSecondaryIndexes": {
            "AlbumTitleIndex": {
```

```
                "CapacityUnits": 0.5
            }
        }
    }
}
```

Para obtener más información, consulte Uso de consultas en DynamoDB en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte Query en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
```

```
 TableName      string
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
```

```
 }
 return movies, err
}
```

Defina una estructura Película para utilizar en este ejemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
```

```
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Consultar una tabla con DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To query items from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedQueryRecords example.
 */
public class Query {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <partitionKeyName> <partitionKeyVal>

                Where:
                    tableName - The Amazon DynamoDB table to put the item in (for
 example, Music3).
                    partitionKeyName - The partition key name of the Amazon
 DynamoDB table (for example, Artist).
                    partitionKeyVal - The value of the partition key that should
 match (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String partitionKeyName = args[1];
        String partitionKeyVal = args[2];

        // For more information about an alias, see:
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.ExpressionAttributeNames.html
        String partitionAlias = "#a";

        System.out.format("Querying %s", tableName);
        System.out.println("");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
```

```
        int count = queryTable(ddb, tableName, partitionKeyName, partitionKeyVal,
 partitionAlias);
        System.out.println("There were " + count + "  record(s) returned");
        ddb.close();
    }

    public static int queryTable(DynamoDbClient ddb, String tableName, String
 partitionKeyName, String partitionKeyVal,
            String partitionAlias) {
        // Set up an alias for the partition key name in case it's a reserved
 word.
        HashMap<String, String> attrNameAlias = new HashMap<String, String>();
        attrNameAlias.put(partitionAlias, partitionKeyName);

        // Set up mapping of the partition name with the value.
        HashMap<String, AttributeValue> attrValues = new HashMap<>();
        attrValues.put(":" + partitionKeyName, AttributeValue.builder()
                .s(partitionKeyVal)
                .build());

        QueryRequest queryReq = QueryRequest.builder()
                .tableName(tableName)
                .keyConditionExpression(partitionAlias + " = :" +
 partitionKeyName)
                .expressionAttributeNames(attrNameAlias)
                .expressionAttributeValues(attrValues)
                .build();

        try {
            QueryResponse response = ddb.query(queryReq);
            return response.count();

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return -1;
    }
}
```

Consultar una tabla con `DynamoDbClient` y un índice secundario.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * Create the Movies table by running the Scenario example and loading the Movie
 * data from the JSON file. Next create a secondary
 * index for the Movies table that uses only the year column. Name the index
 * **year-index**. For more information, see:
 *
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html
 */
public class QueryItemsUsingIndex {
    public static void main(String[] args) {
        String tableName = "Movies";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        queryIndex(ddb, tableName);
        ddb.close();
    }

    public static void queryIndex(DynamoDbClient ddb, String tableName) {
        try {
            Map<String, String> expressionAttributesNames = new HashMap<>();
            expressionAttributesNames.put("#year", "year");
            Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
```

```
            expressionAttributeValues.put(":yearValue",
 AttributeValue.builder().n("2013").build());

            QueryRequest request = QueryRequest.builder()
                    .tableName(tableName)
                    .indexName("year-index")
                    .keyConditionExpression("#year = :yearValue")
                    .expressionAttributeNames(expressionAttributesNames)
                    .expressionAttributeValues(expressionAttributeValues)
                    .build();

            System.out.println("=== Movie Titles ===");
            QueryResponse response = ddb.query(request);
            response.items()
                    .forEach(movie ->
 System.out.println(movie.get("title").s()));

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener información sobre la API, consulte QueryCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
};

docClient.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun queryDynTable(
    tableNameVal: String,
```

```
    partitionKeyName: String,
    partitionKeyVal: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = partitionKeyName

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.S(partitionKeyVal)

    val request =
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
            this.expressionAttributeValues = attrValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        $birthKey = [
            'Key' => [
```

```
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);

    public function query(string $tableName, $key)
    {
        $expressionAttributeValues = [];
        $expressionAttributeNames = [];
        $keyConditionExpression = "";
        $index = 1;
        foreach ($key as $name => $value) {
            $keyConditionExpression .= "#" . array_key_first($value) . " = :v
$index,";
            $expressionAttributeNames["#" . array_key_first($value)] =
 array_key_first($value);
            $hold = array_pop($value);
            $expressionAttributeValues[":v$index"] = [
                array_key_first($hold) => array_pop($hold),
            ];
        }
        $keyConditionExpression = substr($keyConditionExpression, 0, -1);
        $query = [
            'ExpressionAttributeValues' => $expressionAttributeValues,
            'ExpressionAttributeNames' => $expressionAttributeNames,
            'KeyConditionExpression' => $keyConditionExpression,
            'TableName' => $tableName,
        ];
        return $this->dynamoDbClient->query($query);
    }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para PHP.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: invoca una consulta que devuelve elementos de DynamoDB con los valores de SongTitle y Artist especificados.

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

Salida:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Para obtener detalles de la API, consulte Query en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: invoca una consulta que devuelve elementos de DynamoDB con los valores de SongTitle y Artist especificados.

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

Salida:

```
Name                           Value
```

```
----                            -----
Genre                           Country
Artist                          No One You Know
Price                           1.94
CriticRating                    9
SongTitle                       Somewhere Down The Road
AlbumTitle                      Somewhat Famous
```

- Para obtener detalles de la API, consulte Query en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Consultar los elementos mediante una expresión de condición de clave.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
```

```
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
 self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Items"]
```

Consultar los elementos y proyectarlos para devolver un subconjunto de datos.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table
```

```
   def query_and_project_movies(self, year, title_bounds):
        """
        Query for movies that were released in a specified year and that have
titles
        that start within a range of letters. A projection expression is used
        to return a subset of data for each movie.

        :param year: The release year to query.
        :param title_bounds: The range of starting letters to query.
        :return: The list of movies.
        """
        try:
            response = self.table.query(
                ProjectionExpression="#yr, title, info.genres, info.actors[0]",
                ExpressionAttributeNames={"#yr": "year"},
                KeyConditionExpression=(
                    Key("year").eq(year)
                    & Key("title").between(
                        title_bounds["first"], title_bounds["second"]
                    )
                ),
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ValidationException":
                logger.warning(
                    "There's a validation error. Here's the message: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            else:
                logger.error(
                    "Couldn't query for movies. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response["Items"]
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Queries for movies that were released in the specified year.
  #
  # @param year [Integer] The year to query.
  # @return [Array] The list of movies that were released in the specified year.
  def query_items(year)
    response = @table.query(
      key_condition_expression: '#yr = :year',
      expression_attribute_names: { '#yr' => 'year' },
      expression_attribute_values: { ':year' => year }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't query for movies released in #{year}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.items
  end
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Buscar las películas realizadas en el año especificado.

```
pub async fn movies_in_year(
    client: &Client,
    table_name: &str,
    year: u16,
) -> Result<Vec<Movie>, MovieError> {
    let results = client
        .query()
        .table_name(table_name)
        .key_condition_expression("#yr = :yyyy")
        .expression_attribute_names("#yr", "year")
        .expression_attribute_values(":yyyy",
 AttributeValue::N(year.to_string()))
        .send()
        .await?;

    if let Some(items) = results.items {
        let movies = items.iter().map(|v| v.into()).collect();
        Ok(movies)
    } else {
        Ok(vec![])
    }
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    TRY.
        " Query movies for a given year .
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_year }| ) ) ).
        DATA(lt_key_conditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |EQ|
          ) ) ) ).
        oo_result = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_key_conditions ).
        DATA(lt_items) = oo_result->get_items( ).
        "You can loop over the results to get item attributes.
        LOOP AT lt_items INTO DATA(lt_item).
          DATA(lo_title) = lt_item[ key = 'title' ]-value.
          DATA(lo_year) = lt_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_result->get_count( ).
        MESSAGE 'Item count is: ' && lv_count TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Para obtener información sobre la API, consulte Query en la Referencia de la API del AWSSDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB


    /// Get all the movies released in the specified year.
    ///
    /// - Parameter year: The release year of the movies to return.
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = QueryInput(
                expressionAttributeNames: [
                    "#y": "year"
                ],
                expressionAttributeValues: [
                    ":y": .n(String(year))
                ],
                keyConditionExpression: "#y = :y",
                tableName: self.tableName
            )
            // Use "Paginated" to get all the movies.
            // This lets the SDK handle the 'lastEvaluatedKey' property in
 "QueryOutput".

            let pages = client.queryPaginated(input: input)

            var movieList: [Movie] = []
```

```
            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Convert the found movies into `Movie` objects and return an
array
                // of them.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList
        } catch {
            print("ERROR: getMovies:", dump(error))
            throw error
        }
    }
```

- Para obtener detalles de la API, consulte Query en la referencia de la API del SDK de AWS para Swift.

Para obtener más ejemplos de DynamoDB, consulte Ejemplos de código de DynamoDB con los SDK de AWS.

Para crear un índice secundario global para la tabla, continúe en el Paso 6: (opcional) Eliminar la tabla de DynamoDB para limpiar los recursos.

Paso 6: (opcional) Eliminar la tabla de DynamoDB para limpiar los recursos

Si ya no necesita la tabla de Amazon DynamoDB que creó para el tutorial, puede eliminarla. Este paso le permite asegurarse de que no se le cobre por los recursos que no vaya a utilizar. Puede utilizar la consola de DynamoDB o la AWS CLI para eliminar la tabla `Music` que creó en el Paso 1: crear una tabla en DynamoDB.

Para obtener más información sobre las operaciones en DynamoDB, consulte Uso de tablas y datos en DynamoDB.

Consola de administración de AWS

Para eliminar la tabla `Music` mediante la consola:

1. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación izquierdo, elija Tables (Tablas).
3. Elija la casilla de verificación situada junto a la tabla Música en la lista de tablas.
4. Elija Eliminar.

AWS CLI

En el siguiente ejemplo de la AWS CLI, se elimina la tabla `Music` utilizando `delete-table`.

```
aws dynamodb delete-table --table-name Music
```

AWS SDK

En los siguientes ejemplos de código se muestra cómo eliminar una tabla de DynamoDB con un SDK de AWS.

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Deletes a DynamoDB table.
/// </summary>
/// <param name="tableName">The name of the table to delete.</param>
```

```
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
 successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
 deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
 table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
 {tableName}. {ex.Message}");
            return false;
        }
    }
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function dynamodb_delete_table"
    echo "Deletes an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to delete."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
```

```
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho ""

  response=$(aws dynamodb delete-table \
    --table-name "$table_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-table operation failed.$response"
    return 1
  fi

  return 0
}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
```

```
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

################################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
```

```
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información sobre la API, consulte DeleteTable en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Delete an Amazon DynamoDB table.
/*!
  \sa deleteTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteTable(const Aws::String &tableName,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
```

```
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Eliminación de una tabla

En el siguiente ejemplo de `delete-table` se elimina la tabla `MusicCollection`.

```
aws dynamodb delete-table \
    --table-name MusicCollection
```

Salida:

```
{
    "TableDescription": {
        "TableStatus": "DELETING",
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableName": "MusicCollection",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        }
    }
}
```

Para obtener más información, consulte Eliminación de una tabla en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte DeleteTable en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteTable deletes the DynamoDB table and all of its data.
```

```
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
 return err
}
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class DeleteTable {
    public static void main(String[] args) {
```

```
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to delete (for example,
Music3).

                **Warning** This program will delete the table that you specify!
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        System.out.format("Deleting the Amazon DynamoDB table %s...\n",
tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
{
        DeleteTableRequest request = DeleteTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }
```

```
}
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun deleteDynamoDBTable(tableNameVal: String) {
```

```
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    public function deleteTable(string $TableName)
    {
        $this->customWaiter(function () use ($TableName) {
            return $this->dynamoDbClient->deleteTable([
                'TableName' => $TableName,
            ]);
        });
    }
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para PHP.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: elimina la tabla especificada. Se le solicitará una confirmación antes de continuar con la operación.

```
Remove-DDBTable -TableName "myTable"
```

Ejemplo 2: elimina la tabla especificada. No se le solicitará una confirmación antes de continuar con la operación.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Para obtener detalles de la API, consulte DeleteTable en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: elimina la tabla especificada. Se le solicitará una confirmación antes de continuar con la operación.

```
Remove-DDBTable -TableName "myTable"
```

Ejemplo 2: elimina la tabla especificada. No se le solicitará una confirmación antes de continuar con la operación.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Para obtener detalles de la API, consulte DeleteTable en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
```

```
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Deletes the table.
  def delete_table
    @table.delete
    @table = nil
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete table. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
```

```
  end
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
pub async fn delete_table(client: &Client, table: &str) ->
 Result<DeleteTableOutput, Error> {
    let resp = client.delete_table().table_name(table).send().await;

    match resp {
        Ok(out) => {
            println!("Deleted table");
            Ok(out)
        }
        Err(e) => Err(Error::Unhandled(e.into())),
    }
}
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    TRY.
        lo_dyn->deletetable( iv_tablename = iv_table_name ).
        " Wait till the table is actually deleted.
        lo_dyn->get_waiter( )->tablenotexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'Table ' && iv_table_name && ' deleted.' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table ' && iv_table_name && ' does not exist' TYPE 'E'.
      CATCH /aws1/cx_dynresourceinuseex.
        MESSAGE 'The table cannot be deleted since it is in use' TYPE 'E'.
    ENDTRY.
```

- Para obtener información sobre la API, consulte DeleteTable en la Referencia de la API del AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB
```

```
    ///
    /// Deletes the table from Amazon DynamoDB.
    ///
    func deleteTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteTableInput(
                tableName: self.tableName
            )
            _ = try await client.deleteTable(input: input)
        } catch {
            print("ERROR: deleteTable:", dump(error))
            throw error
        }
    }
```

- Para obtener información sobre la API, consulte DeleteTable en la Referencia de la API de AWS SDK para Swift.

Para obtener más ejemplos de DynamoDB, consulte Ejemplos de código de DynamoDB con los SDK de AWS.

Continuación del aprendizaje de DynamoDB

Para obtener más información sobre el uso de Amazon DynamoDB, consulte los siguientes temas:

- Uso de tablas y datos en DynamoDB
- Uso de elementos y atributos en DynamoDB
- Consulta de tablas en DynamoDB
- Uso de índices secundarios globales en DynamoDB
- Trabajo con transacciones
- Aceleración en memoria con DynamoDB Accelerator (DAX)
- Programación con DynamoDB y los SDK de AWS

Generación de código de infraestructura para Amazon DynamoDB mediante Consola a código

La característica Consola a código de Amazon Q Developer simplifica la administración de la infraestructura de Amazon DynamoDB al transformar los pasos de creación manual de tablas en código de automatización reproducible. Esta capacidad ayuda a los desarrolladores a escalar de manera eficiente la configuración de los recursos de la base de datos en sus entornos. Para obtener más información, consulte Automatización de Servicios de AWS con Consola a código de Amazon Q Developer.

Consola a código captura configuraciones detalladas de tablas de DynamoDB, incluidas las claves de partición, las claves de clasificación, la configuración de rendimiento aprovisionada y los índices secundarios, y las convierte en plantillas precisas de infraestructura como código. Al utilizar la IA generativa, la herramienta garantiza que el código generado mantenga la compatibilidad de parámetros establecida durante el flujo de trabajo de la consola.

Los desarrolladores pueden generar código de infraestructura de DynamoDB en varios formatos, como:

- AWS Cloud Development Kit (AWS CDK) en TypeScript, Python y Java
- CloudFormation en YAML o JSON

Este método permite a los equipos:

- Estandarización de la administración de recursos de bases de datos
- Implementación de una infraestructura controlada por versiones
- Reducción de los errores de configuración manual

Consola a código para Amazon DynamoDB está disponible en todas las regiones de AWS comerciales y ofrece una solución eficaz para transformar las configuraciones manuales en código de infraestructura automatizado y reproducible.

Funcionamiento

Cuando se utiliza Consola a código con DynamoDB, el proceso suele implicar:

1. Creación de prototipos en la consola: utilice la consola de DynamoDB para crear y configurar recursos, como tablas. Consulte Conectarse a Amazon DynamoDB para obtener más información.
2. Grabación de acciones: Consola a código graba estas acciones a medida que se realizan.
3. Generación de código: la característica utiliza las capacidades de IA generativa de Amazon Q Developer para transformar las acciones de la consola en código reutilizable en el formato que prefiera.
4. Personalización del código: a continuación, puede copiar o descargar este código y personalizarlo aún más para sus cargas de trabajo de producción.

Beneficios del uso de Consola a código con DynamoDB

Automatización simplificada

Convierta la creación y configuración manuales de tablas de DynamoDB en código reutilizable con un solo clic.

Prácticas recomendadas

El código generado sigue las prácticas recomendadas guiadas de AWS para lograr implementaciones fiables.

Puente entre la consola y el código

No tiene que elegir entre usar Consola de administración de AWS o la infraestructura como código (IaC). Puede utilizar ambos enfoques juntos.

Desarrollo acelerado

Comience rápidamente con el código de automatización que se puede personalizar aún más para su uso en producción.

Ejemplos de casos de uso

- Creación de tablas de DynamoDB con atributos, claves y ajustes de capacidad específicos
- Configuración de índices secundarios globales e índices secundarios locales
- Configuración de políticas de escalado automático para tablas de DynamoDB
- Establecimiento de configuraciones de copia de seguridad y restauración
- Creación y administración de DynamoDB Streams

Introducción

Para empezar a usar Consola a código con DynamoDB:

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/DynamoDB/.
2. Comience a crear o modificar los recursos de DynamoDB a través de la interfaz de la consola.
3. Use la característica Consola a código a fin de generar código para las acciones en el formato que prefiera.
4. Copie o descargue el código generado y personalícelo según sus necesidades específicas.

Para obtener más información e instrucciones detalladas sobre cómo utilizar Consola a código, consulte Automatización de Servicios de AWS con Consola a código de Amazon Q Developer en la Guía del usuario de Amazon Q Developer.

Funcionamiento de Amazon DynamoDB

En las secciones siguientes se incluye información general sobre los componentes del servicio de Amazon DynamoDB y las interacciones entre ellos.

Temas

- Hoja de referencia para DynamoDB
- Componentes básicos de Amazon DynamoDB
- API de DynamoDB
- Tipos de datos y reglas de nomenclatura admitidos en Amazon DynamoDB
- Clases de tablas de DynamoDB
- Particiones y distribución de datos en DynamoDB
- Información sobre cómo pasar de SQL a NoSQL
- Recursos y herramientas de aprendizaje de Amazon DynamoDB

Hoja de referencia para DynamoDB

Esta hoja de referencia proporciona una base rápida para trabajar con Amazon DynamoDB y los distintos AWS SDK.

Configuración inicial

1. Regístrese en AWS.
2. Obtenga una clave de acceso de AWS para acceder a DynamoDB mediante programación.
3. Configure las credenciales de DynamoDB.

Véase también:

- Configuración de DynamoDB (servicio web)
- Introducción a DynamoDB
- Información básica de los componentes principales

SDK o CLI

Elija el SDK preferido o configure la AWS CLI.

> **Note**
>
> Cuando se utiliza la AWS CLI en Windows, una barra invertida (\) que no esté entre comillas se considera un retorno de carro. Además, debe usar escape entre comillas y corchetes dentro de otras comillas. Para ver un ejemplo, consulte la pestaña Windows en "Create a table" (Crear una tabla) en la siguiente sección.

Véase también:

- AWS CLI con DynamoDB
- Introducción a DynamoDB: paso 2

Acciones básicas

En esta sección se proporciona código para las tareas básicas de DynamoDB. Para obtener más información sobre estas tareas, consulte Introducción a DynamoDB y los AWS SDK.

Creación de una tabla

Default

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --table-class STANDARD
```

Windows

```
aws dynamodb create-table ^
    --table-name Music ^
```

```
    --attribute-definitions ^
        AttributeName=Artist,AttributeType=S ^
        AttributeName=SongTitle,AttributeType=S ^
    --key-schema AttributeName=Artist,KeyType=HASH
AttributeName=SongTitle,KeyType=RANGE ^
    --billing-mode PAY_PER_REQUEST ^
    --table-class STANDARD
```

Escribir elemento en una tabla

```
aws dynamodb put-item \ --table-name Music \ --item file://item.json
```

Leer elemento de una tabla

```
aws dynamodb get-item \ --table-name Music \ --item file://item.json
```

Eliminar un elemento de una tabla

```
aws dynamodb delete-item --table-name Music --key file://key.json
```

Consulta de una tabla

```
aws dynamodb query --table-name Music
--key-condition-expression "ArtistName=:Artist and SongName=:Songtitle"
```

Eliminación de una tabla

```
aws dynamodb delete-table --table-name Music
```

Mostrar nombres de tablas

```
aws dynamodb list-tables
```

Reglas de nomenclatura

- Todos los nombres se deben codificar mediante UTF-8 y distinguen entre mayúsculas y minúsculas.

- Los nombres de las tablas y los índices deben tener entre 3 y 255 caracteres, que solo pueden ser los siguientes:
  - `a-z`
  - `A-Z`
  - `0-9`
  - `_` (guion bajo)
  - `-` (guion)
  - `.` (punto)
- Los nombres de atributo deben tener al menos un carácter y un tamaño inferior a 64 KB.

Para obtener más información, consulte Naming rules (Reglas de nomenclatura).

Conceptos básicos sobre Service Quotas

Unidades de lectura y escritura

- Unidad de capacidad de lectura (RCU): una lectura altamente coherente por segundo o dos lecturas coherentes posteriores por segundo, para elementos con un tamaño de hasta 4 KB.
- Unidad de capacidad de escritura (WCU): una escritura por segundo para los elementos con un tamaño de hasta 1 KB.

Límites de tabla

- Tamaño de tabla: no existe ningún límite práctico del tamaño de una tabla. Las tablas no presentan restricciones en cuanto al número de elementos o de bytes.
- Número de tablas: para cualquier cuenta de AWS, existe una cuota inicial de 2500 tablas por región de AWS.
- Límite de tamaño de página para consulta y análisis: hay un límite de 1 MB por página, por consulta o análisis. Si los parámetros de la consulta o la operación de análisis de una tabla arrojan más de 1 MB de datos, DynamoDB devuelve los elementos coincidentes iniciales. También devuelve una propiedad `LastEvaluatedKey` que puede utilizar en una nueva solicitud para leer la página siguiente.

Índices

- Índices secundarios locales (LSI): puede definir un máximo de cinco índices secundarios locales. Los LSI son útiles principalmente cuando un índice debe tener una gran coherencia con la tabla base.
- Índices secundarios globales (GSI): existe una cuota predeterminada de 20 índices secundarios globales por tabla.
- Atributos de índice secundario proyectado por tabla: puede proyectar un máximo de 100 atributos en todos los índices secundarios locales y globales de una tabla. Esto solo se aplica a los atributos proyectados especificados por el usuario.

Claves de partición

- La longitud mínima de un valor de clave de partición es de 1 byte. La longitud máxima es de 2048 bytes.
- No existe ningún límite práctico respecto al número de valores diferentes de clave de partición, ni para tablas ni para los índices secundarios.
- La longitud mínima de un valor de clave de ordenación es de 1 byte. La longitud máxima es de 1024 bytes.
- En general, no existe ningún límite práctico respecto al número de valores diferentes de clave de ordenación por cada valor de clave de partición. Hay una excepción en las tablas que utilizan índices secundarios.

Para obtener más información sobre los índices secundarios, el diseño de la clave de partición y el diseño de la clave de clasificación, consulte Prácticas recomendadas.

Límites para los tipos de datos de uso común

- Cadena: la longitud de una cadena está limitada por el tamaño de elemento máximo de 400 KB. Los valores de tipo String son Unicode con codificación binaria UTF-8.
- Número: un número puede tener hasta 38 dígitos de precisión y puede ser positivo, negativo o cero.
- Binario: la longitud de un valor binario está limitada por el tamaño de elemento máximo de 400 KB. Las aplicaciones que utilizan atributos binarios deben codificar los datos en cifrado base64 antes de enviarlos a DynamoDB.

Para obtener una lista completa de los tipos de datos admitidos, consulte Data types (Tipos de datos). Para obtener más información, consulte también Service Quotas.

Elementos, atributos y parámetros de expresión

El tamaño máximo de un elemento en DynamoDB es de 400 KB, que incluye la longitud en formato binario de los nombres de los atributos (longitud en UTF-8) y las longitudes en formato binario de los valores de los atributos (longitud en UTF-8). El nombre de los atributos se tiene en cuenta al calcular el límite de tamaño.

No existe ningún límite en el número de valores de una lista, un mapa o un conjunto, siempre y cuando el elemento que contiene los valores se ajuste al límite de tamaño de elemento de 400 KB.

Para parámetros de expresión, la longitud máxima de cualquier cadena de expresión es 4 KB.

Para obtener más información sobre el tamaño del elemento, los atributos y los parámetros de expresión, consulte Service Quotas.

Más información

- Seguridad
- Monitoreo y registro
- Uso de secuencias
- Copias de seguridad y recuperación a un momento dado
- Integración con otros servicios de AWS
- Referencia de la API del
- Centro de arquitectura: prácticas recomendadas de bases de datos
- Tutoriales de vídeo
- Foro de DynamoDB

Componentes básicos de Amazon DynamoDB

En DynamoDB se trabaja principalmente con tablas, elementos y atributos. Una tabla es una colección de elementos y cada elemento es una colección de atributos. DynamoDB utiliza claves principales para identificar de forma exclusiva cada elemento de una tabla. Puede utilizar DynamoDB Streams para capturar los eventos de modificación de datos en las tablas de DynamoDB.

En DynamoDB se aplican algunos límites. Para obtener más información, consulte Cuotas en Amazon DynamoDB.

En el siguiente vídeo encontrará una introducción a las tablas, los elementos y los atributos.

Tablas, elementos y atributos

Tablas, elementos y atributos

People

Primary key Partition key: PersonID	Attributes			
101	LastName	FirstName	Phone	
	Smith	Fred	555-4321	
102	LastName	FirstName	Address	
	Jones	Mary	{"Street":"123 Main","City":"Anytown","State":"OH","ZIPCode":12345}	
103	LastName	FirstName	FavoriteColor	Address
	Stephens	Howard	Blue	{"Street":"123 Main","City":"London","PostalCode":"ER3 5K8"}

A continuación, se indican los componentes básicos de DynamoDB:

- Tablas: al igual que otros sistemas de base de datos, DynamoDB almacena datos en tablas. Una tabla es una colección de datos. Por ejemplo, consulte la tabla de ejemplo denominada People, que puede utilizar para almacenar información de contacto personal sobre amigos, familiares u otras personas de interés. También podría utilizar una tabla Cars para almacenar información sobre los vehículos que conducen las personas.
- Elementos: Cada tabla contiene cero o más elementos. Un elemento es un grupo de atributos que puede identificarse de forma exclusiva entre todos los demás elementos. En una tabla People, cada elemento representa a una persona. En una tabla Cars, cada elemento representa un vehículo. Los elementos de DynamoDB son similares en muchos aspectos a las filas, los registros o las tuplas de otros sistemas de bases de datos. En DynamoDB, no existe ningún límite respecto al número de elementos que pueden almacenarse en una tabla.

- Atributos: cada elemento se compone de uno o varios atributos. Un atributo es un componente fundamental de los datos, que no es preciso dividir más. Por ejemplo, un elemento de una tabla Personas contiene los atributos PersonID, LastName, FirstName, etc. En una tabla Departamento, un elemento podría tener atributos tales como DepartmentID, Nombre, Mánager, etc. En DynamoDB, los atributos se parecen en muchos aspectos a los campos o columnas en otros sistemas de bases de datos.

En el siguiente diagrama se muestra una tabla denominada People que contiene algunos ejemplos de elementos y atributos.

```
People

{
    "PersonID": 101,
    "LastName": "Smith",
    "FirstName": "Fred",
    "Phone": "555-4321"
}

{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
                "Street": "123 Main",
                "City": "Anytown",
                "State": "OH",
                "ZIPCode": 12345
    }
}

{
    "PersonID": 103,
    "LastName": "Stephens",
    "FirstName": "Howard",
    "Address": {
                "Street": "123 Main",
                "City": "London",
                "PostalCode": "ER3 5K8"
    },
    "FavoriteColor": "Blue"
```

```
}
```

Tenga en cuenta lo siguiente en relación con la tabla People:

- Cada elemento de la tabla tiene un identificador único, o clave principal, que lo distingue de todos los demás. En la tabla People, la clave principal consta de un atributo (PersonID).
- Dejando a un lado la clave principal, la tabla People no tiene esquema. Esto significa que no es preciso definir de antemano los atributos ni sus tipos de datos. Cada elemento puede tener sus propios atributos diferentes.
- La mayoría de los atributos son escalares, lo que significa que solo pueden tener un valor. Las cadenas y los números son ejemplos comunes de escalares.
- Algunos de los elementos tienen un atributo anidado (Address). DynamoDB admite atributos anidados hasta un máximo de 32 niveles de profundidad.

A continuación se muestra otro ejemplo de tabla denominada Music que podría utilizar para llevar un registro de una colección de música.

```
Music

{
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot",
    "AlbumTitle": "Hey Now",
    "Price": 1.98,
    "Genre": "Country",
    "CriticRating": 8.4
}

{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The Road",
    "AlbumTitle": "Somewhat Famous",
    "Genre": "Country",
    "CriticRating": 8.4,
    "Year": 1984
}

{
    "Artist": "The Acme Band",
```

```
    "SongTitle": "Still in Love",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 2.47,
    "Genre": "Rock",
    "PromotionInfo": {
        "RadioStationsPlaying": [
            "KHCR",
            "KQBX",
            "WTNR",
            "WJJH"
        ],
        "TourDates": {
            "Seattle": "20150622",
            "Cleveland": "20150630"
        },
        "Rotation": "Heavy"
    }
}

{
    "Artist": "The Acme Band",
    "SongTitle": "Look Out, World",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 0.99,
    "Genre": "Rock"
}
```

Tenga en cuenta lo siguiente en relación con la tabla Music:

- La clave principal de Music consta de dos atributos (Artist y SongTitle). Cada elemento de la tabla debe tener estos dos atributos. La combinación de Artist y SongTitle distingue a cada elemento de la tabla de todos los demás.
- Dejando a un lado la clave principal, la tabla Music no tiene esquema. Esto significa que no es preciso definir de antemano los atributos ni sus tipos de datos. Cada elemento puede tener sus propios atributos diferentes.
- Uno de los elementos tiene un atributo anidado (PromotionInfo), que contiene otros atributos anidados. DynamoDB admite atributos anidados hasta un máximo de 32 niveles de profundidad.

Para obtener más información, consulte Uso de tablas y datos en DynamoDB.

Clave principal

Al crear una tabla, además de asignarle un nombre, debe especificar su clave principal. La clave principal identifica de forma única a cada elemento de la tabla, de manera que no puede haber dos elementos con la misma clave.

DynamoDB admite dos tipos distintos de clave principal:

- Clave de partición: una clave principal simple que consta de un solo atributo denominado clave de partición.

  DynamoDB utiliza el valor de clave de partición como información de entrada a una función hash interna. El resultado de la función hash determina la partición (almacenamiento físico interno de DynamoDB) donde se almacenará el elemento.

  En una tabla que solo tiene una clave de partición, no puede haber dos elementos que tengan el mismo valor de clave de partición.

  La tabla People descrita en Tablas, elementos y atributos es un ejemplo de una tabla con una clave principal simple (PersonID). Puede obtener acceso a cualquier elemento de la tabla People directamente al proporcionar el valor de PersonId de dicho elemento.
- Clave de partición y clave de ordenación: este tipo de clave se denomina clave principal compuesta y consta de dos atributos. El primer atributo es la clave de partición y el segundo, la clave de ordenación.

  DynamoDB utiliza el valor de clave de partición como información de entrada a una función hash interna. El resultado de la función hash determina la partición (almacenamiento físico interno de DynamoDB) donde se almacenará el elemento. Todos los elementos con el mismo valor de clave de partición se almacenan en posiciones contiguas, ordenados según el valor de la clave de ordenación.

  En una tabla que tenga una clave de partición y una clave de ordenación, es posible que varios elementos tengan el mismo valor de clave de partición. Sin embargo, esos elementos deben tener valores de clave de ordenación distintos.

  La tabla Music descrita en Tablas, elementos y atributos es un ejemplo de una tabla con una clave principal compuesta (Artist y SongTitle). Puede obtener acceso a cualquier elemento de la tabla Music directamente al proporcionar los valores de Artist y SongTitle de dicho elemento.

Una clave principal compuesta ofrece más flexibilidad a la hora de consultar datos. Por ejemplo, si proporciona el valor de Artist, DynamoDB recupera todas las canciones de ese intérprete. Para recuperar solo un subconjunto de canciones de un intérprete determinado, proporcione un valor de Artist y un intervalo de valores de SongTitle.

> **Note**
>
> La clave de partición de un elemento también se denomina atributo hash. El término atributo hash alude al uso de una función hash interna en DynamoDB para distribuir los elementos de datos de manera uniforme entre las particiones, según sus valores de clave de partición.
> La clave de clasificación de un elemento también se denomina atributo de rango. El término atributo de intervalo alude al hecho de que DynamoDB almacena en ubicaciones físicamente contiguas todos los elementos que tienen la misma clave de partición, ordenados según el valor de la clave de ordenación.

Cada atributo de clave principal debe ser escalar (es decir, solo puede contener un único valor). Los únicos tipos de datos que se permiten para los atributos de clave principal son String, Number y Binary. A los demás atributos sin clave no se les aplican restricciones de esta índole.

Índices secundarios

Puede crear uno o varios índices secundarios en una tabla. Un índice secundario le permite consultar los datos de la tabla usando una clave alternativa, además de realizar consultas basadas en la clave principal. DynamoDB no requiere que se usen índices; sin embargo, estos ofrecen a las aplicaciones mayor flexibilidad a la hora de consultar los datos. Después de crear un índice secundario en una tabla, podrá leer los datos en el índice prácticamente de la misma forma que en la tabla.

DynamoDB admite dos tipos de índices:

- Índice secundario global: índice con una clave de partición y una clave de ordenación que pueden diferir de las claves de la tabla. Los valores de clave principal en índices secundarios globales no tienen que ser únicos.
- Índice secundario local: índice que tiene la misma clave de partición que la tabla, pero una clave de ordenación distinta.

En DynamoDB, los índices secundarios globales (GSI) son índices que abarcan toda la tabla y permiten realizar consultas en todas las claves de partición. Los índices secundarios locales (LSI) son índices que tienen la misma clave de partición que la tabla base, pero una clave de clasificación diferente.

Cada tabla de DynamoDB tiene una cuota de 20 índices secundarios globales (cuota predeterminada) y 5 índices secundarios locales.

En el ejemplo de la tabla Música mostrado anteriormente, puede consultar los elementos de datos por Artista (clave de partición) o por Artista y SongTitle (claves de partición y clasificación). ¿Qué sucede si también desea consultar los datos por género musical (Genre) y título de álbum (AlbumTitle)? Para ello, puede crear un índice basado en Genre y AlbumTitle y, a continuación, consultarlo prácticamente de la misma forma que se consultaría la tabla Music.

El siguiente diagrama muestra la tabla Music de ejemplo, con un nuevo índice llamado GenreAlbumTitle. En el índice, Genre es la clave de partición y AlbumTitle es la clave de ordenación.

Tabla Music	GenreAlbumTitle

<pre>{
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot",
    "AlbumTitle": "Hey Now",
    "Price": 1.98,
    "Genre": "Country",
    "CriticRating": 8.4
}</pre>
	<pre>{
	    "Genre": "Country",
	    "AlbumTitle": "Hey Now",
	    "Artist": "No One You Know",
	    "SongTitle": "My Dog Spot"
	}</pre>

<pre>{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The
 Road",
    "AlbumTitle": "Somewhat Famous",
    "Genre": "Country",
    "CriticRating": 8.4,
    "Year": 1984
}</pre>
	<pre>{
	    "Genre": "Country",
	    "AlbumTitle": "Somewhat Famous",
	    "Artist": "No One You Know",
	    "SongTitle": "Somewhere Down The
	 Road"
	}</pre>

Tabla Music	GenreAlbumTitle

<pre>{
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love",
    "AlbumTitle": "The Buck Starts
 Here",
    "Price": 2.47,
    "Genre": "Rock",
    "PromotionInfo": {
        "RadioStationsPlaying": {
            "KHCR",
            "KQBX",
            "WTNR",
            "WJJH"
        },
        "TourDates": {
            "Seattle": "20150622",
            "Cleveland": "20150630"
        },
        "Rotation": "Heavy"
    }
}</pre>
	<pre>{
	    "Genre": "Rock",
	    "AlbumTitle": "The Buck Starts
	 Here",
	    "Artist": "The Acme Band",
	    "SongTitle": "Still In Love"
	}</pre>

<pre>{
    "Artist": "The Acme Band",
    "SongTitle": "Look Out, World",
    "AlbumTitle": "The Buck Starts
 Here",
    "Price": 0.99,
    "Genre": "Rock"
}</pre>
	<pre>{
	    "Genre": "Rock",
	    "AlbumTitle": "The Buck Starts
	 Here",
	    "Artist": "The Acme Band",
	    "SongTitle": "Look Out, World"
	}</pre>

Tenga en cuenta lo siguiente en relación con el índice GenreAlbumTitle:

- Cada índice pertenece a una tabla, que se denomina la tabla base del índice. En el ejemplo anterior, Music es la tabla base del índice GenreAlbumTitle.
- DynamoDB mantiene los índices automáticamente. Al agregar, actualizar o eliminar un elemento de la tabla base, DynamoDB agrega, actualiza o elimina el elemento correspondiente en los índices que pertenecen a dicha tabla.
- Al crear un índice, se especifica qué atributos de la tabla base se copiarán, o proyectarán, en el índice. Como mínimo, DynamoDB proyecta en el índice los atributos de clave de la tabla base. Esto es lo que sucede con el índice `GenreAlbumTitle`, en el que únicamente se proyectan los atributos de clave de la tabla `Music`.

Puede consultar el índice GenreAlbumTitle para encontrar todos los álbumes de un género musical determinado (por ejemplo, todos los álbumes de música Rock). También puede consultar el índice para encontrar todos los álbumes de un determinado género musical que tengan un título de álbum específico (por ejemplo, todos los álbumes de música Country cuyo título comience por la letra H).

Para obtener más información, consulte Mejora del acceso con índices secundarios en DynamoDB.

DynamoDB Streams

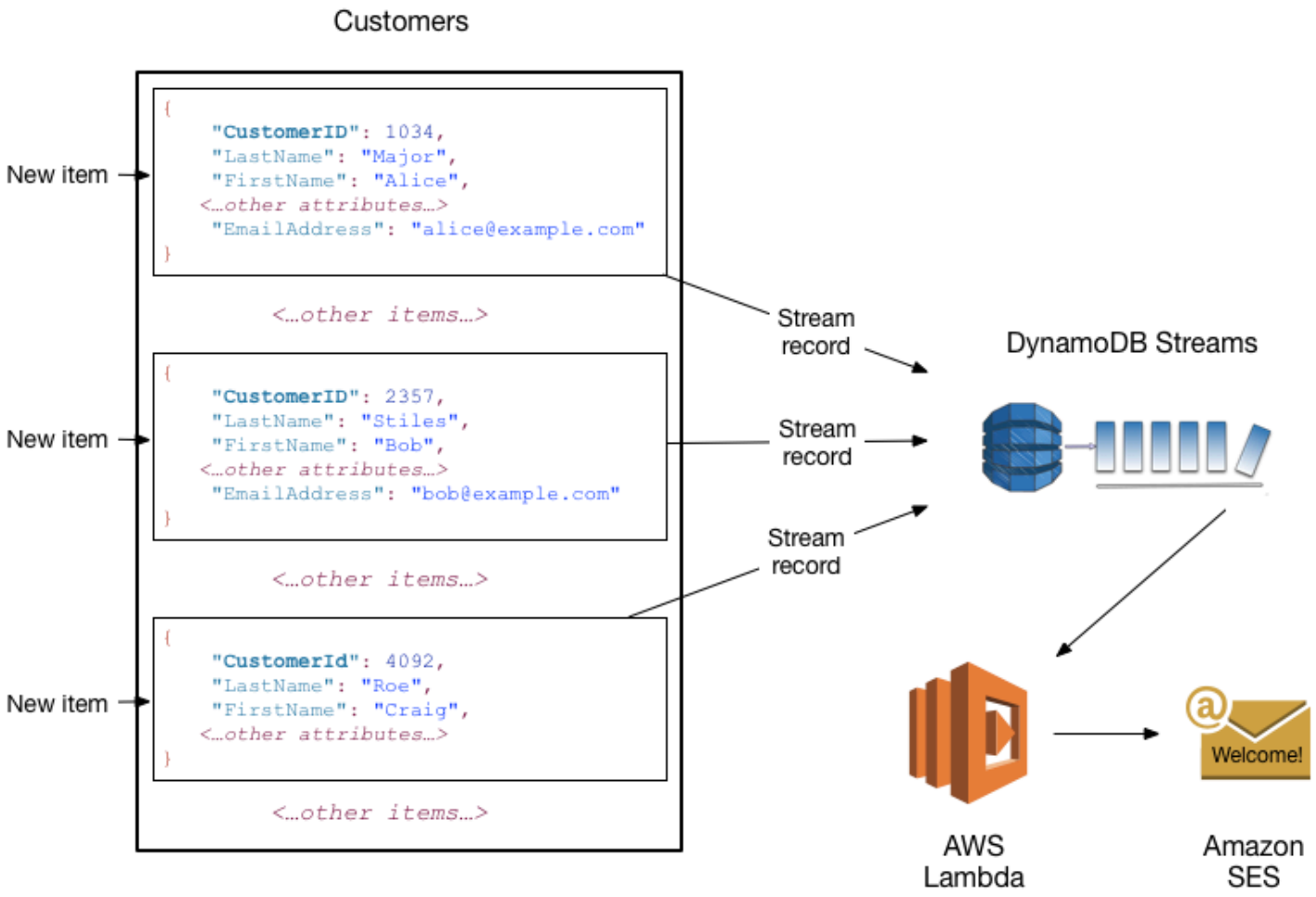
DynamoDB Streams es una característica opcional que captura los eventos de modificación de datos que se producen en las tablas de DynamoDB. Los datos de estos eventos aparecen en la secuencia prácticamente en tiempo real y en el orden en que se han producido.

Cada evento se representa mediante un registro de secuencia. Si habilita una secuencia en una tabla, DynamoDB Streams escribe un registro de secuencia cada vez que se produzcan los siguientes eventos:

- Se agrega un nuevo elemento a la tabla: la secuencia captura una imagen del elemento completo, incluidos todos sus atributos.
- Se actualiza un elemento: la secuencia captura las imágenes de "antes" y "después" de los atributos del elemento que se han modificado.
- Se elimina un elemento de la tabla: la secuencia captura una imagen del elemento completo antes de eliminarlo.

Cada registro de secuencia también contiene el nombre de la tabla, la marca temporal del evento y otros metadatos. Los registros de secuencia tienen una vida útil de 24 horas; después, se eliminan automáticamente de la secuencia.

Puede utilizar DynamoDB Streams conjuntamente con AWS Lambda para crear un desencadenador; es decir, un código que se ejecute automáticamente cada vez que aparezca un evento de interés en una transmisión. Por ejemplo, tomemos una tabla Customers que contiene la información de clientes de una compañía. Supongamos que desea enviar un mensaje de correo electrónico de bienvenida a cada nuevo cliente. Podría habilitar una transmisión en esa tabla y, a continuación, asociar la transmisión con una función Lambda. La función Lambda se ejecutaría cada vez que apareciese un nuevo registro en la transmisión, pero solamente procesaría los nuevos elementos agregados a la tabla Customers. Para cualquier elemento que tuviera el atributo `EmailAddress`, la función Lambda podría invocar a Amazon Simple Email Service (Amazon SES) para que enviase un mensaje de correo electrónico a esa dirección.



> **Note**
>
> En este ejemplo, observe que el último cliente, Craig Roe, no recibirá un mensaje de correo electrónico, porque no tiene el atributo `EmailAddress`.

Además de los disparadores, DynamoDB Streams permite utilizar soluciones sumamente eficientes, tales como replicación de datos en el seno de regiones de AWS y entre ellas, vistas materializadas de datos en las tablas de DynamoDB o análisis de datos mediante vistas materializadas de Kinesis, entre otras.

Para obtener más información, consulte Captura de datos de cambios para DynamoDB Streams.

API de DynamoDB

Para trabajar con Amazon DynamoDB, la aplicación debe utilizar algunas operaciones de la API sencillas. A continuación se ofrece un resumen de estas operaciones, organizadas por categorías.

> **Note**
>
> Para obtener una lista completa de las operaciones de la API, consulte la Referencia de la API de Amazon DynamoDB.

Temas

- Plano de control
- Plano de datos
- DynamoDB Streams
- Transacciones

Plano de control

Las operaciones del plano de control permiten crear y administrar tablas de DynamoDB. También permiten usar índices, secuencias y otros objetos que dependen de las tablas.

- `CreateTable`: crea una nueva tabla. Si lo prefiere, puede crear uno o varios índices secundarios y habilitar DynamoDB Streams para la tabla.
- `DescribeTable`: devuelve información sobre una tabla, como su esquema de clave principal, ajustes de rendimiento e información de índice.
- `ListTables`: devuelve los nombres de todas las tablas en una lista.
- `UpdateTable`: modifica los ajustes de una tabla o sus índices, crea o elimina nuevos índices en una tabla o modifica los ajustes de DynamoDB Streams de una tabla.

- `DeleteTable`: elimina de DynamoDB una tabla y todos los objetos que dependen de ella.

Plano de datos

Las operaciones del plano de datos permiten llevar a cabo acciones de creación, lectura, actualización y eliminación (también denominadas operaciones CRUD, por sus siglas en inglés) con los datos de una tabla. Algunas de las operaciones del plano de datos permiten también leer datos de un índice secundario.

Puede utilizar PartiQL: un lenguaje de consulta compatible con SQL para Amazon DynamoDB, para realizar estas operaciones CRUD o puede usar las API CRUD clásicas de DynamoDB que separa cada operación en una llamada a API distinta.

PartiQL: lenguaje de consulta compatible con SQL

- `ExecuteStatement`: lee varios elementos de una tabla. También puede escribir o actualizar un solo elemento de una tabla. Al escribir o actualizar un solo elemento, debe especificar los atributos de clave principal.
- `BatchExecuteStatement`: escribe, actualiza o lee varios elementos de una tabla. Resulta más eficiente que `ExecuteStatement`, porque la aplicación solo tiene que completar un único recorrido de ida y vuelta para escribir o leer todos los elementos.

API clásicas

Creación de datos

- `PutItem`: escribe un solo elemento en una tabla. Debe especificar los atributos de clave principal, pero no es preciso especificar otros atributos.
- `BatchWriteItem`: escribe hasta 25 elementos en una tabla. Resulta más eficiente que llamar a `PutItem` varias veces, porque la aplicación solo tiene que completar un único recorrido de ida y vuelta para escribir todos los elementos.

Lectura de datos

- `GetItem`: recupera un solo elemento de una tabla. Es preciso especificar la clave principal del elemento que se desea recuperar. Puede recuperar la totalidad del elemento o solo un subconjunto de sus atributos.

- `BatchGetItem`: recupera hasta 100 elementos de una o varias tablas. Resulta más eficiente que llamar a `GetItem` varias veces, porque la aplicación solo tiene que completar un único recorrido de ida y vuelta para leer todos los elementos.
- `Query`: recupera todos los elementos que tienen una clave de partición determinada. Debe especificar el valor de clave de partición. Puede recuperar la totalidad de los elementos o solo un subconjunto de sus atributos. De forma opcional, puede aplicar una condición a los valores de la clave de ordenación de tal forma que solo se recupere un subconjunto de los datos que tienen la misma clave de partición. Puede utilizar esta operación en una tabla, siempre y cuando esta tenga tanto una clave de partición como una clave de ordenación. También puede utilizar esta operación en un índice, siempre y cuando este tenga tanto una clave de partición como una clave de ordenación.
- `Scan`: recupera todos los elementos de la tabla o el índice especificados. Puede recuperar la totalidad de los elementos o solo un subconjunto de sus atributos. Si lo desea, puede aplicar una condición de filtrado para devolver solamente aquellos valores que le interesan y descartar los demás.

Actualización de datos

- `UpdateItem`: modifica uno o varios atributos de un elemento. Es preciso especificar la clave principal del elemento que se desea modificar. Puede agregar nuevos atributos, así como modificar o eliminar los existentes. También puede realizar actualizaciones condicionales, de tal forma que la actualización solamente se lleve a cabo cuando se cumpla una condición definida por el usuario. Si lo desea, puede implementar un contador atómico, que incrementa o disminuye el valor de un atributo numérico sin interferir con las demás solicitudes de escritura.

Eliminación de datos

- `DeleteItem`: elimina un solo elemento de una tabla. Es preciso especificar la clave principal del elemento que se desea eliminar.
- `BatchWriteItem`: elimina hasta 25 elementos de una o varias tablas. Resulta más eficiente que llamar a `DeleteItem` varias veces, porque la aplicación solo tiene que completar un único recorrido de ida y vuelta para eliminar todos los elementos.

> **Note**
>
> Puede usar `BatchWriteItem` para crear y eliminar datos.

DynamoDB Streams

Las operaciones de DynamoDB Streams permiten habilitar o deshabilitar una secuencia en una tabla y obtener acceso a los registros de modificación de datos contenidos en una secuencia.

- `ListStreams`: devuelve una lista de todas las secuencias o solamente la secuencia de una tabla concreta.
- `DescribeStream`: devuelve información sobre una secuencia, como su nombre de recurso de Amazon (ARN) y sobre dónde puede comenzar la aplicación a leer los primeros registros de la secuencia.
- `GetShardIterator`: devuelve un iterador de fragmentos, que es una estructura de datos que la aplicación utiliza para recuperar los registros de la secuencia.
- `GetRecords`: recupera uno o varios registros de secuencia mediante el iterador de fragmentos especificado.

Transacciones

Las transacciones proporcionan atomicidad, coherencia, aislamiento y durabilidad (ACID, atomicity, consistency, isolation, and durability), lo que le permite mantener la exactitud de los datos en sus aplicaciones más fácilmente.

Puede utilizar PartiQL: un lenguaje de consulta compatible con SQL para Amazon DynamoDB, para realizar operaciones transaccionales o puede usar las API CRUD clásicas de DynamoDB que separa cada operación en una llamada a API distinta.

PartiQL: lenguaje de consulta compatible con SQL

- `ExecuteTransaction`: operación por lote que permite operaciones CRUD para varios elementos dentro y distintas tablas con un resultado garantizado de “todo o nada”.

API clásicas

- `TransactWriteItems`— Una operación por lote que permite operaciones `Put`, `Update` y `Delete` con varios elementos dentro y distintas tablas con un resultado garantizado de “todo o nada”.
- `TransactGetItems`— Una operación por lote que permite `Get` para recuperar varios elementos de una o varias tablas.

Tipos de datos y reglas de nomenclatura admitidos en Amazon DynamoDB

En esta sección se describen las reglas de nomenclatura de Amazon DynamoDB y los distintos tipos de datos que DynamoDB admite. Existen límites que se aplican a los tipos de datos. Para obtener más información, consulte Tipos de datos.

Temas

- Reglas de nomenclatura
- Tipos de datos
- Descriptores de tipos de datos

Reglas de nomenclatura

Las tablas, los atributos y otros objetos de DynamoDB deben tener nombres. Los nombres deben ser significativos y concisos; por ejemplo, Products, Books y Authors son nombres que indican claramente su significado.

A continuación, se indican las reglas de nomenclatura de DynamoDB:

- Todos los nombres deben codificarse mediante UTF-8 y distinguen entre mayúsculas y minúsculas.
- Los nombres de las tablas y los índices deben tener entre 3 y 255 caracteres, que solo pueden ser los siguientes:
  - `a-z`
  - `A-Z`
  - `0-9`

- _ (guion bajo)
- - (guion)
- . (punto)

- Los nombres de atributo deben tener al menos un carácter y un tamaño inferior a 64 KB. Se considera una práctica recomendada mantener los nombres de los atributos lo más cortos posible. Esto contribuye a reducir las unidades de solicitud de lectura consumidas, ya que los nombres de los atributos se incluyen en la medición del uso del almacenamiento y del rendimiento.

  A continuación se muestran las excepciones. Los siguientes nombres de atributo no pueden tener más de 255 caracteres:

  - Nombres de clave de partición de índice secundario
  - Nombres de clave de clasificación de índice secundario
  - Nombres de atributos proyectados especificados por el usuario (aplicables solo a índices secundarios locales)

Palabras reservadas y caracteres especiales

DynamoDB tiene una lista de palabras reservadas y caracteres especiales. Para ver una lista completa, consulte Palabras reservadas en DynamoDB. También los siguientes caracteres tienen un significado especial en DynamoDB: # (hash) y : (dos puntos).

Aunque DynamoDB permite utilizar estas palabras reservadas y caracteres especiales en los nombres, recomendamos que evite hacerlo, porque tendría que definir variables de marcador de posición cada vez que utilizase estos nombres en una expresión. Para obtener más información, consulte Nombres de atributos de expresión (Alias) en DynamoDB.

Tipos de datos

DynamoDB admite muchos tipos de datos para los atributos de una tabla. Se pueden categorizar como se indica a continuación:

- Tipos escalares: un tipo escalar es aquel que puede representar exactamente un valor. Los tipos escalares son Number, String, Binary, Boolean y Null.
- Tipos de documentos: un tipo de documento puede representar una estructura compleja con atributos anidados, como los que se encontraría en un documento JSON. Los tipos de documentos son List y Map.

- Tipos de conjuntos: un tipo de conjunto puede representar varios valores escalares. Los tipos de conjuntos son String Set, Number Set y Binary Set.

Al crear una tabla o un índice secundario, debe especificar los nombres y los tipos de datos de cada uno de los atributos de clave principal (clave de partición y clave de ordenación). Además, cada atributo de clave principal debe definirse como de tipo String, Number o Binary.

DynamoDB es una base de datos NoSQL sin esquema. Esto significa que, a excepción de los atributos de clave principal, no tiene que definir atributos ni tipos de datos cuando crea las tablas. En comparación, las bases de datos relacionales requieren que se definan los nombres y los tipos de datos de cada columna al crear la tabla.

A continuación se muestran las descripciones de cada tipo de datos, junto con ejemplos en formato JSON.

Tipos escalares

Los tipos escalares son Number, String, Binary, Boolean y Null.

Número

Los números pueden ser positivos, negativos o cero. Los números pueden tener hasta 38 dígitos de precisión. Si excede este tope, se generará una excepción. Si necesita una precisión superior a 38 dígitos, puede utilizar cadenas.

- Rango de valores positivos: de 1E-130 a 9.99999999999999999999999999999999999999E+125
- Rango de valores negativos: de -9.99999999999999999999999999999999999999E+125 a -1E-130

En DynamoDB, los números se representan con longitud variable. Los ceros iniciales y finales se recortan.

Todos los números se envían a través de la red a DynamoDB como cadenas, para maximizar la compatibilidad entre lenguajes y bibliotecas. Sin embargo, DynamoDB los trata como atributos de tipo Number en las operaciones matemáticas.

Puede utilizar el tipo de datos Number para representar una fecha o una marca temporal. Una forma de hacerlo es utilizar fechas en formato de tiempo Unix, es decir, el número de segundos desde las 00:00:00 UTC del 1 de enero de 1970. Por ejemplo, la fecha de inicio `1437136300` representa las 12:31:40 UTC del 17 de julio de 2015.

Para obtener más información, consulte http://en.wikipedia.org/wiki/Unix_time.

Cadena

Los valores de tipo String son Unicode con codificación binaria UTF-8. La longitud mínima de una cadena puede ser cero si el atributo no se utiliza como clave de un índice o tabla y tiene un límite de tamaño máximo de 400 KB para los elementos de DynamoDB.

Las siguientes restricciones adicionales son aplicables a atributos de clave principal definidos como cadena de tipo:

- Para una clave principal simple, la longitud máxima del valor del primer atributo (clave de partición) es 2048 KB.
- Para una clave principal compuesta, la longitud máxima del valor del segundo atributo (clave de ordenación) es 1024 KB.

DynamoDB recopila y compara las cadenas utilizando los bytes de la codificación de cadena UTF-8 subyacente. Por ejemplo, "a" (0x61) es mayor que "A" (0x41) y "¿" (0xC2BF) es mayor que "z" (0x7A).

Puede utilizar el tipo de datos String para representar una fecha o una marca temporal. Una forma de hacerlo es utilizar cadenas ISO 8601, tal y como se muestra en estos ejemplos:

- `2016-02-15`
- `2015-12-21T17:42:34Z`
- `20150311T122706Z`

Para obtener más información, consulte http://en.wikipedia.org/wiki/ISO_8601.

> **Note**
>
> A diferencia de las bases de datos relacionales convencionales, DynamoDB no admite de forma nativa un tipo de datos de fecha y hora. En cambio, puede ser útil almacenar los datos de fecha y hora como un tipo de dato numérico, mediante el tiempo de época de Unix.

Binario

Los atributos de tipo Binary pueden almacenar cualquier tipo de datos binarios, como texto comprimidos, datos cifrados o imágenes. Siempre que DynamoDB compara valores de tipo Binary, trata cada byte de los datos binarios como sin signo.

La longitud de un atributo binario puede ser cero si el atributo no se utiliza como clave de un índice o tabla y tiene un límite de tamaño máximo de 400 KB para los elementos de DynamoDB.

Si define un atributo de clave principal como atributo de tipo Binary, se aplican las siguientes restricciones adicionales:

- Para una clave principal simple, la longitud máxima del valor del primer atributo (clave de partición) es 2048 KB.
- Para una clave principal compuesta, la longitud máxima del valor del segundo atributo (clave de ordenación) es 1024 KB.

Las aplicaciones deben codificar los valores de tipo Binary en formato codificado en base64 antes de enviarlos a DynamoDB. Al recibirlos, DynamoDB decodifica los datos y los convierte a matrices de bytes sin signo; a continuación, utiliza ese resultado como longitud del atributo de tipo Binary.

El siguiente ejemplo es un atributo de tipo Binary en el que se utiliza texto con la codificación en base64.

```
dGhpcyB0ZXh0IGlzIGJhc2U2NC1lbmNvZGVk
```

Booleano

Un atributo de tipo Boolean puede almacenar los valores `true` o `false`.

Nulo

Null representa un atributo con un estado desconocido o sin definir.

Tipos de documentos

Los tipos de documentos son List y Map. Estos tipos de datos pueden anidarse unos en otros para representar estructuras de datos complejas con un máximo de 32 niveles de profundidad.

No existe ningún límite respecto al número de valores de una lista o un mapa, siempre y cuando el elemento que contenga los valores se ajuste al límite de tamaño de elemento de DynamoDB (400 KB).

El valor de un atributo puede ser una cadena vacía o un valor binario vacío si el atributo no se utiliza en una tabla o clave de índice. El valor de un atributo no puede ser un conjunto vacío (conjunto de cadenas, conjunto de números o conjunto binario); sin embargo, sí se permiten listas y mapas vacíos. Se pueden utilizar valores binarios y de cadena vacíos en las listas y los mapas. Para obtener más información, consulte Atributos.

Enumeración

Un atributo de tipo List puede almacenar una colección ordenada de valores. Las listas deben ir entre corchetes: .: `[ ... ]`

Una lista es similar a una matriz JSON. No hay ninguna limitación respecto a los tipos de datos que se pueden almacenar en una entrada de lista y no es preciso que las entradas de una entrada de lista sean del mismo tipo.

En el siguiente ejemplo se muestra una lista que contiene dos cadenas y un número.

```
FavoriteThings: ["Cookies", "Coffee", 3.14159]
```

> **Note**
>
> DynamoDB permite usar entradas individuales contenidas en las listas, aunque estas entradas estén anidadas a gran profundidad. Para obtener más información, consulte Uso de expresiones en DynamoDB.

Asignación

Un atributo de tipo Map puede almacenar una colección desordenada de pares nombre-valor. Los mapas deben ir entre llaves: .: `{ ... }`

Un mapa es similar a un objeto JSON. No hay ninguna limitación respecto a los tipos de datos que se pueden almacenar en una entrada de mapa y no es preciso que las entradas de un mapa sean del mismo tipo.

Los mapas son idóneos para almacenar documentos JSON en DynamoDB. En el siguiente ejemplo se muestra un mapa que contiene una cadena, un número y una lista anidada que contiene otro mapa.

```
{
    Day: "Monday",
    UnreadEmails: 42,
    ItemsOnMyDesk: [
        "Coffee Cup",
        "Telephone",
        {
            Pens: { Quantity : 3},
            Pencils: { Quantity : 2},
            Erasers: { Quantity : 1}
        }
    ]
}
```

> **Note**
>
> DynamoDB permite usar entradas individuales contenidas en los mapas, aunque estas entradas estén anidadas a gran profundidad. Para obtener más información, consulte Uso de expresiones en DynamoDB.

Sets

DynamoDB admite tipos que representan conjuntos de valores de tipo number, string o binary. Todas las entradas de un conjunto deben ser del mismo tipo. Por ejemplo, un conjunto de números solo puede contener números y un conjunto de cadenas solo puede contener cadenas.

No existe ningún límite respecto al número de valores de un conjunto, siempre y cuando el elemento que contenga los valores se ajuste al límite de tamaño de elemento de DynamoDB (400 KB).

Cada valor contenido en un conjunto debe ser único. No se conserva el orden de los valores dentro de un conjunto. Por lo tanto, sus aplicaciones no deben confiar en ningún orden particular de elementos dentro del conjunto. DynamoDB no admite conjuntos vacíos; sin embargo, sí se pueden utilizar valores binarios y de cadena vacíos dentro de un conjunto.

En el siguiente ejemplo se muestra un conjunto de cadenas, un conjunto de números y un conjunto de valores binarios:

```
["Black", "Green", "Red"]

[42.2, -19, 7.5, 3.14]

["U3Vubnk=", "UmFpbnk=", "U25vd3k="]
```

Descriptores de tipos de datos

El protocolo de bajo nivel de la API de DynamoDB utiliza descriptores de tipo de datos como tokens que indican a DynamoDB cómo interpretar cada atributo.

A continuación se muestra una lista completa de descriptores de tipos de datos de DynamoDB:

- **`S`** – cadena
- **`N`** – número
- **`B`** – Binary
- **`BOOL`** – booleano
- **`NULL`** – Null
- **`M`** – mapa
- **`L`** – List
- **`SS`** – String Set
- **`NS`** – Number Set
- **`BS`** – Binary Set

Clases de tablas de DynamoDB

DynamoDB ofrece dos clases de tablas diseñadas para ayudarle a optimizar los costos. La clase de tabla DynamoDB Estándar es la predeterminada y se recomienda para la gran mayoría de las cargas de trabajo. La clase de tabla DynamoDB Estándar - Acceso poco frecuente (DynamoDB Standard-IA) está optimizada para tablas en las que el almacenamiento es el costo dominante. Por ejemplo, las tablas que almacenan datos a los que se accede con poca frecuencia, como registros de aplicaciones, publicaciones antiguas en redes sociales, historial de pedidos de comercio electrónico y antiguos logros en juegos, son buenas candidatas para la clase de tabla Standard-IA. Consulte Precio de Amazon DynamoDB para obtener más información sobre los precios.

Cada tabla DynamoDB está asociada a una clase de tabla (DynamoDB Estándar de forma predeterminada). Todos los índices secundarios asociados a la tabla utilizan la misma clase de tabla. Cada clase de tabla ofrece diferentes precios para el almacenamiento de datos, así como para las solicitudes de lectura y escritura. Puede seleccionar la clase de tabla más rentable para usted en función de sus patrones de uso de almacenamiento y rendimiento.

La elección de una clase de tabla no es permanente. Puede cambiar esta configuración mediante la Consola de administración de AWS, la CLI de AWS o el SDK de AWS. DynamoDB también permite administrar la clase de tabla mediante AWS CloudFormation para las tablas de una región concreta y las tablas globales. Para obtener más información acerca de cómo seleccionar una clase de tabla, consulte Aspectos que tener en cuenta a la hora de elegir una clase de tabla en DynamoDB.

Particiones y distribución de datos en DynamoDB

Amazon DynamoDB almacena los datos en las particiones. Una partición es una asignación de almacenamiento a una tabla que se sustenta en discos de estado sólido (SSD) y se replica automáticamente en varias zonas de disponibilidad de una región de AWS. DynamoDB se encarga de todo lo que concierne a la administración de las particiones para que usted no tenga que ocuparse de ello.

Al crear una tabla, su estado inicial es `CREATING`. Durante esta fase, DynamoDB asigna particiones suficientes a la tabla para que pueda satisfacer los requisitos de rendimiento aprovisionado. Puede comenzar a escribir y leer datos en la tabla una vez que su estado haya cambiado a `ACTIVE`.

DynamoDB asigna particiones adicionales a una tabla en las siguientes situaciones:

- Si aumenta los ajustes de rendimiento aprovisionado de la tabla de tal forma que se supere el límite admitido por las particiones existentes.
- Si una partición existente se llena al límite de su capacidad y se requiere más espacio de almacenamiento.

La administración de las particiones tiene lugar automáticamente en segundo plano y es transparente para las aplicaciones. La tabla permanece disponible a lo largo del proceso y responde en todo momento a los requisitos de rendimiento aprovisionado.

Para obtener más información, consulta Diseño de claves de partición.

Los índices secundarios globales de DynamoDB también constan de particiones. Los datos de un índice secundario global se almacenan de forma independiente de los datos de la tabla base, pero las particiones de índices se comportan prácticamente igual que las particiones de tablas.

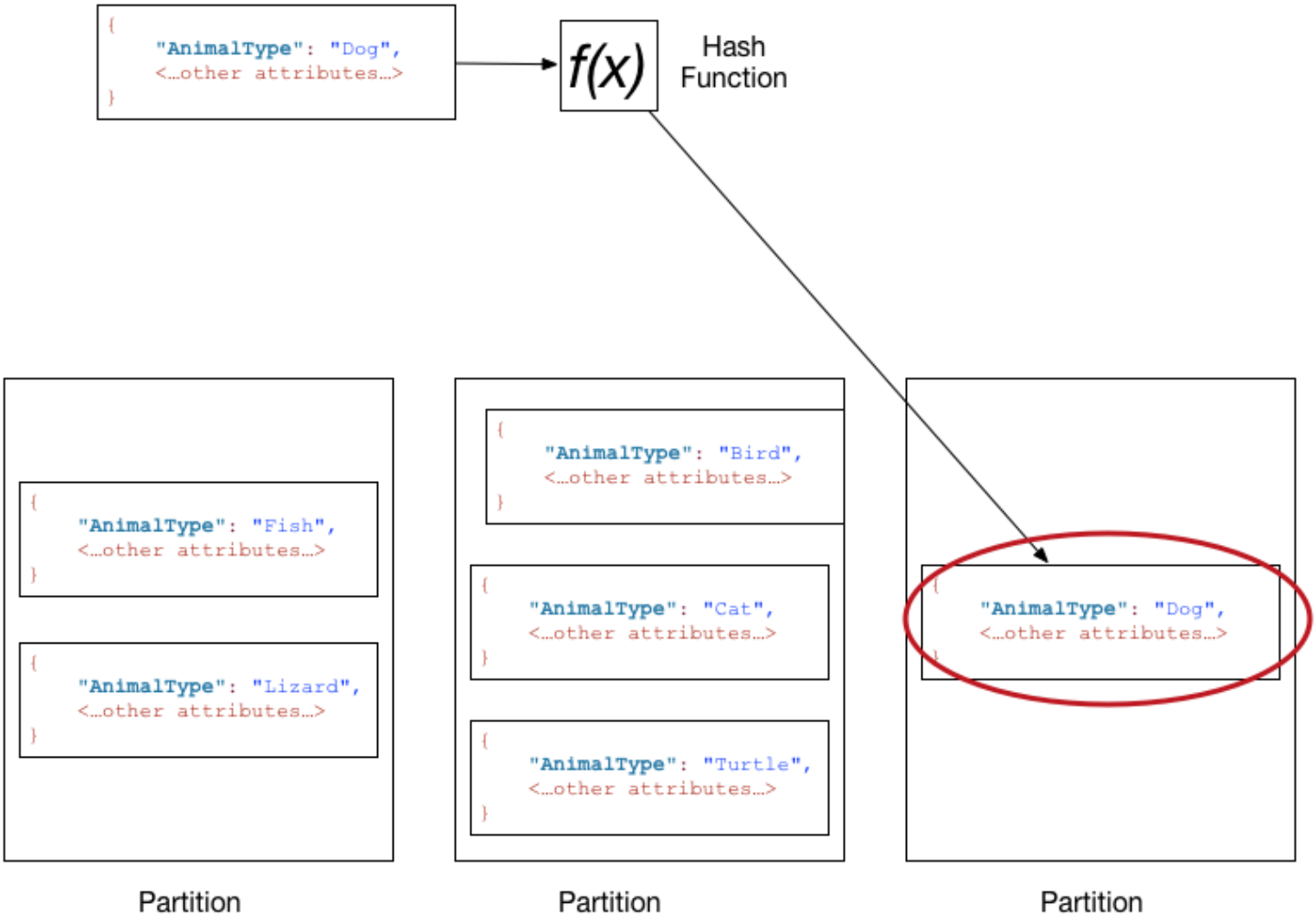
Distribución de datos: clave de partición

Si la tabla tiene una clave principal simple (solo clave de partición), DynamoDB almacenará y recuperará cada elemento basándose en su valor de clave de partición.

Para escribir un elemento en la tabla, DynamoDB utiliza el valor de la clave de partición como información de entrada para una función hash interna. El valor del resultado de la función hash determina la partición donde se almacenará el elemento.

Para leer un elemento de la tabla, debe especificar el valor de clave de partición del elemento. DynamoDB utiliza este valor como información de entrada para la función hash para obtener la partición en la que se encuentra el elemento.

En el siguiente diagrama se muestra una tabla denominada Pets que abarca varias particiones. La clave principal de la tabla es AnimalType (solo se muestra este atributo de clave). DynamoDB utiliza la función hash para determinar dónde se almacenará un nuevo elemento, en este caso de acuerdo con el valor hash de la cadena Dog. Tenga en cuenta que los elementos no se almacenan de forma ordenada. La ubicación de cada elemento viene determinada por el valor hash de su clave de partición.



> **Note**
>
> DynamoDB está optimizado para distribuir los elementos uniformemente entre las particiones de una tabla, con independencia del número de particiones que haya. Recomendamos elegir una clave de partición que pueda tener un amplio abanico de valores distintos en relación con el número de elementos de la tabla.

Distribución de datos: clave de partición y clave de clasificación

Si la tabla tiene una clave principal compuesta (clave de partición y clave de ordenación), DynamoDB calcula el valor hash de la clave de partición de la misma forma que se describe en Distribución de datos: clave de partición. Sin embargo, tiende a mantener los elementos que tienen el mismo valor de clave de partición juntos y ordenados según el valor del atributo de la clave de clasificación. Al conjunto de elementos que tienen el mismo valor de la clave de partición se le denomina recopilación

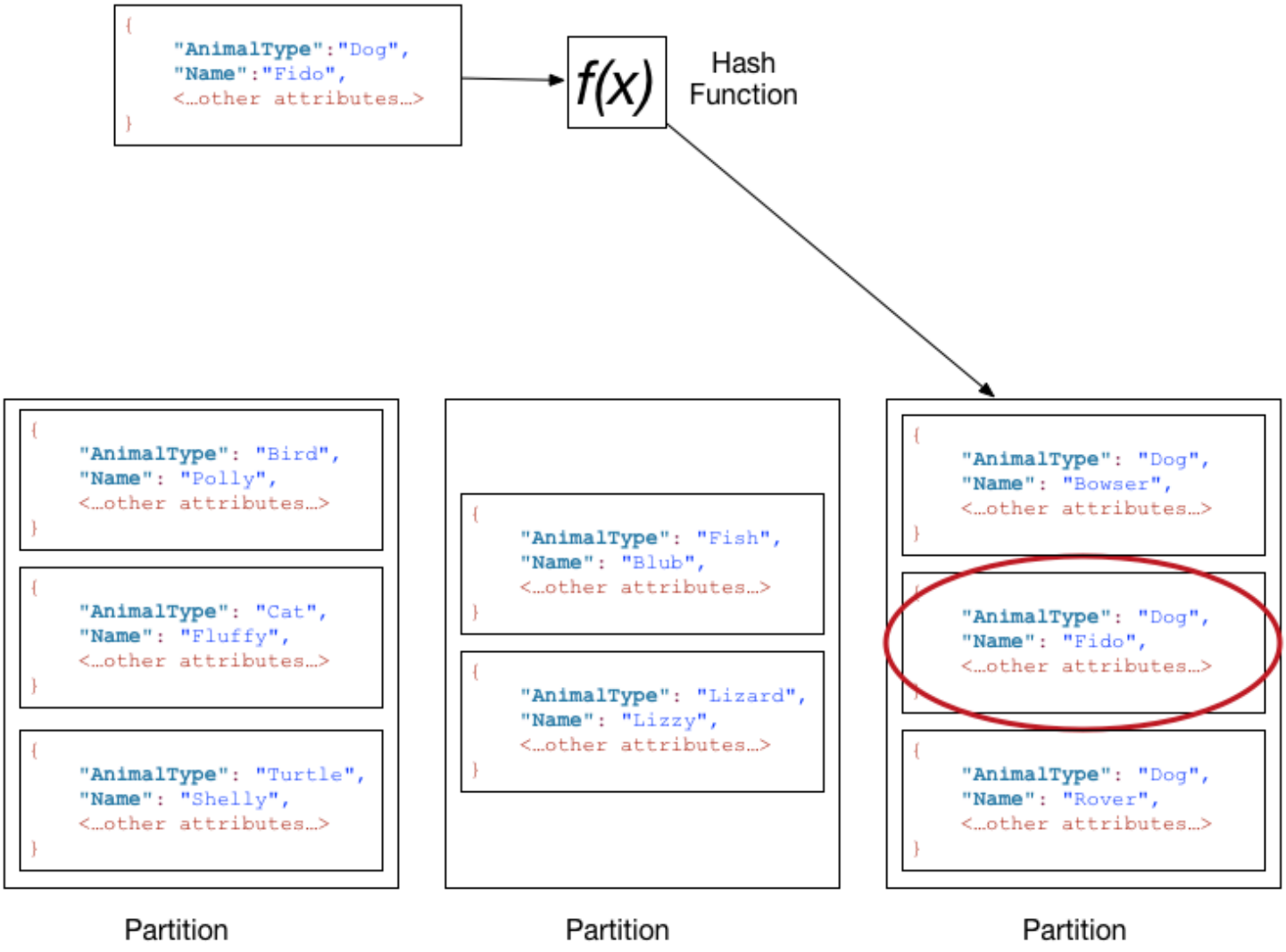
de elementos. Las recopilaciones de elementos están optimizadas para recuperar de forma eficiente los rangos de los elementos de la recopilación. Si la tabla no tiene índices secundarios locales, DynamoDB dividirá automáticamente la recopilación de elementos en tantas particiones como sea necesario para almacenar los datos y garantizar el rendimiento de lectura y escritura.

Para escribir un elemento en la tabla, DynamoDB calcula el valor hash de su clave de partición, con el fin de determinar qué partición debe contener el elemento. En dicha partición, varios elementos podrían tener el mismo valor de clave de partición. Por tanto, DynamoDB almacena el elemento con los otros que tienen la misma clave de partición, en orden ascendente por clave de ordenación.

Para leer un elemento de la tabla, debe especificar los valores de sus claves de partición y ordenación. DynamoDB calcula el valor hash de la clave de partición para obtener la partición en la que se encuentra el elemento.

Puede leer varios elementos de la tabla en una misma operación (`Query`), si los elementos que desee leer tengan el mismo valor de clave de partición. DynamoDB devuelve todos los elementos que contienen ese valor de clave de partición. De forma opcional, puede aplicar una condición a la clave de ordenación para que devuelva solamente los elementos comprendidos en un rango de valores determinado.

Supongamos que la tabla Pets tiene una clave principal compuesta que consta de AnimalType (clave de partición) y Name (clave de ordenación). En el siguiente diagrama se ilustra lo que ocurre cuando DynamoDB escribe un elemento con un valor de clave de partición Dog y un valor de clave de ordenación Fido.



Para leer el mismo elemento de la tabla Pets, DynamoDB calcula el valor hash de Dog para obtener la partición en la que están almacenados estos elementos. A continuación, DynamoDB examina los valores de los atributos de clave de ordenación hasta que encuentra Fido.

Para leer todos los elementos cuyo valor de AnimalType es Dog, puede emitir una operación `Query` sin especificar una condición de clave de ordenación. De forma predeterminada, los elementos se devuelven en el orden en el que están almacenados (es decir, en orden ascendente, según su clave de ordenación). Si lo prefiere, puede solicitar que se muestren por orden descendente.

Para consultar solo algunos de los elementos Dog, puede aplicar una condición a la clave de ordenación (por ejemplo, solo los elementos Dog cuyo valor de Name comience por una letra comprendida entre la A y la K).

> **Note**
>
> En una tabla de DynamoDB, no existe ningún límite superior respecto al número de valores diferentes de clave de ordenación por cada valor de clave de partición. Si necesita almacenar muchos miles de millones de elementos Dog en la tabla Pets, DynamoDB asignaría almacenamiento suficiente para satisfacer este requisito automáticamente.

Información sobre cómo pasar de SQL a NoSQL

Si es desarrollador de aplicaciones, es posible que tenga alguna experiencia con el sistema de administración de bases de datos relacionales (RDBMS, por sus siglas en inglés) y con el lenguaje de consulta estructurada SQL. Cuando comience a utilizar Amazon DynamoDB, observará numerosas similitudes, pero también bastantes diferencias. El término NoSQL se utiliza para describir los sistemas de bases de datos no relacionales que tienen un alto grado de disponibilidad y escalabilidad y están optimizados para ofrecer un rendimiento elevado. En lugar del modelo relacional, las bases de datos NoSQL (como DynamoDB) utilizan modelos alternativos de administración de datos, como los pares clave-valor o el almacenamiento de documentos. Para obtener más información, consulte ¿Qué es NoSQL?.

Amazon DynamoDB admite PartiQL, un lenguaje de consulta de código abierto compatible con SQL que facilita la consulta de datos de forma eficiente, independientemente de dónde o en qué formato se almacenen. Con PartiQL, puede procesar fácilmente datos estructurados de bases de datos relacionales, datos semiestructurados y anidados en formatos de datos abiertos e incluso datos sin esquema en bases de datos NoSQL o de documentos que permiten distintos atributos para diferentes filas. Para obtener más información, consulte Lenguaje de consulta PartiQL.

En las siguientes secciones se describen las tareas que suelen llevarse a cabo con las bases de datos y se comparan y contrastan las instrucciones de SQL con las operaciones de DynamoDB equivalentes.

> **Note**
>
> Los ejemplos de SQL de esta sección son compatibles con el RDBMS MySQL.
> En los ejemplos de DynamoDB de esta sección, se muestra el nombre de la operación de DynamoDB junto con los parámetros de dicha operación en formato JSON.

Temas

- Elección entre relacional (SQL) y NoSQL
- Diferencias al acceder a una base de datos relacional (SQL) y DynamoDB
- Diferencias entre una base de datos relacional (SQL) y DynamoDB al crear una tabla
- Diferencias entre obtener información de tablas de una base de datos relacional (SQL) y DynamoDB
- Diferencias entre una base de datos relacional (SQL) y DynamoDB al escribir datos en una tabla
- Diferencias entre una base de datos relacional (SQL) y DynamoDB al leer datos de una tabla
- Diferencias entre una base de datos relacional (SQL) y DynamoDB al administrar índices
- Diferencias entre una base de datos relacional (SQL) y DynamoDB al modificar datos de una tabla
- Diferencias entre una base de datos relacional (SQL) y DynamoDB al eliminar datos de una tabla
- Diferencias entre una base de datos relacional (SQL) y DynamoDB al eliminar una tabla

Elección entre relacional (SQL) y NoSQL

Los requisitos de las aplicaciones actuales son más exigentes que nunca. Por ejemplo, un juego online podría comenzar con unos pocos usuarios y una pequeña cantidad de datos. No obstante, si el juego tiene éxito, puede superar fácilmente los recursos del sistema de administración de bases de datos subyacente. Es frecuente que las aplicaciones basadas en Web tengan cientos, miles o millones de usuarios simultáneos, que generen nuevos datos del orden de terabytes o más. Las bases de datos de este tipo de aplicaciones deben administrar decenas o cientos de miles de lecturas y escrituras por segundo.

Amazon DynamoDB es apropiada para cargas de trabajo de este tipo. Como desarrollador, puede comenzar a pequeña escala y aumentar gradualmente a medida que su aplicación adquiera popularidad. DynamoDB se escala de manera fluida hasta administrar enormes cantidades de datos y de usuarios.

Para obtener más información sobre el modelado tradicional de bases de datos relacionales y cómo adaptarlo a DynamoDB, consulte Prácticas recomendadas para modelar datos relacionales en DynamoDB.

En la siguiente tabla se muestran algunas diferencias generales nivel entre un sistema de administración de bases de datos relacionales (RDBMS) y DynamoDB.

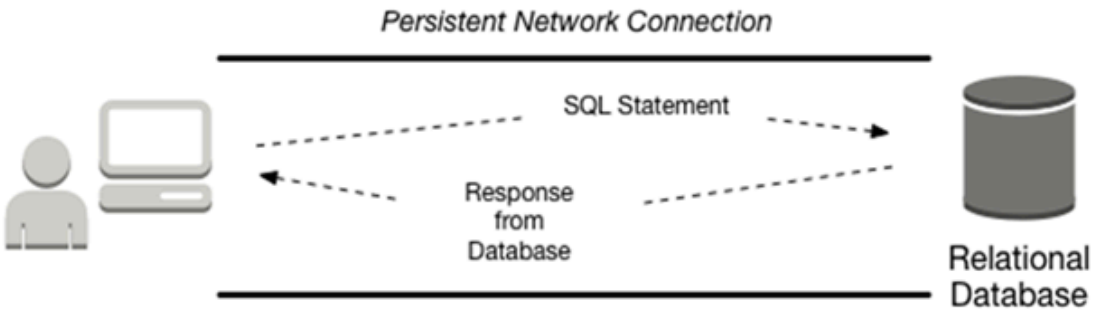
Característica	Sistema de base de datos relacional (RDBMS)	Amazon DynamoDB
Cargas de trabajo óptimas	Consultas ad-hoc; almacenamiento de datos; OLAP (procesamiento analítico online).	Aplicaciones a escala Web, tales como redes sociales, juegos, intercambio de contenido multimedia o Internet de las cosas (IoT).
Modelo de datos	El modelo relacional requiere un esquema bien definido, cuyos datos estén normalizados en tablas, filas y columnas. Además, se definen todas las relaciones entre las tablas, las columnas, los índices y otros componentes de las bases de datos.	DynamoDB no utiliza ningún esquema. Cada tabla debe tener una clave principal para identificar cada elemento de datos de forma exclusiva, pero no existen restricciones similares para otros atributos que no son de clave. DynamoDB puede administrar datos estructurados o semiestructurados, incluidos los documentos JSON.
Acceso a los datos	SQL es el lenguaje de consulta estándar para almacenar y recuperar datos. Las bases de datos relacionales ofrecen un amplio conjunto de herramientas que simplifican el desarrollo de las aplicaciones basadas en bases de datos, pero en todas estas herramientas se utiliza SQL.	Puede utilizar Consola de administración de AWS, AWS CLI o NoSQL WorkBench para trabajar con DynamoDB y llevar a cabo tareas ad-hoc. PartiQL, un lenguaje de consulta compatible con SQL, le permite seleccionar, insertar, actualizar y eliminar datos en DynamoDB. Las aplicaciones pueden usar los kits de desarrollo de software (SDK) de AWS para trabajar con DynamoDB a través de interfaces basadas en objetos,

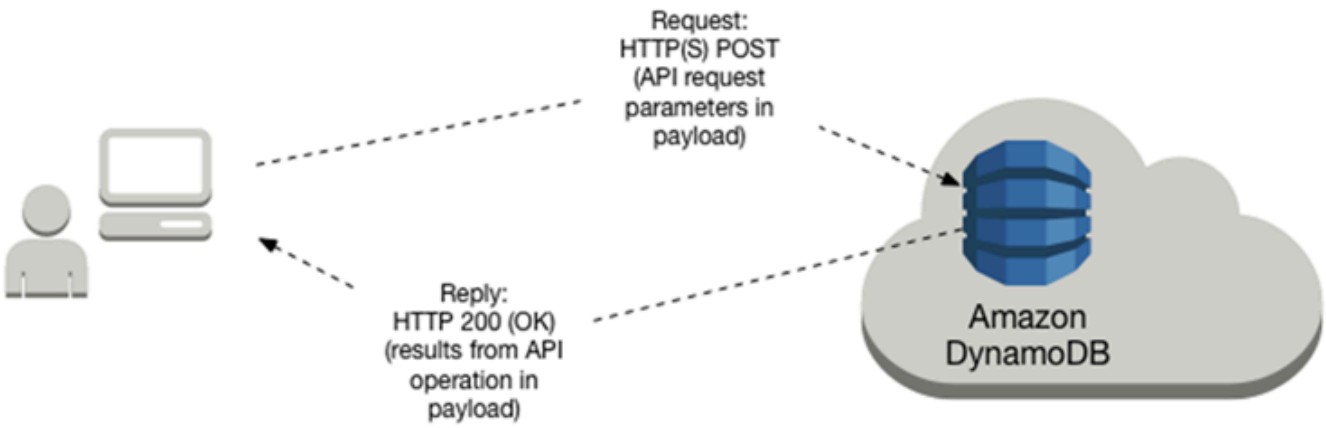
Característica	Sistema de base de datos relacional (RDBMS)	Amazon DynamoDB
		centradas en documentos o de bajo nivel.
Desempeño	Las bases de datos relacionales están optimizadas para el almacenamiento, de modo que, por lo general, su rendimiento depende del subsistema de disco. Los desarrolladores y administradores de bases de datos deben optimizar las consultas, los índices y la estructura de las tablas para lograr el máximo rendimiento.	DynamoDB está optimizado para la informática, por lo que el rendimiento depende principalmente del hardware subyacente y de la latencia de la red. Al tratarse de un servicio administrado, DynamoDB desvincula al desarrollador y a sus aplicaciones de estos detalles de implementación, para que pueda centrarse en diseñar y crear aplicaciones robustas de alto rendimiento.
Escalado	Resulta más fácil de escalar con un hardware más rápido. Además, es posible que las tablas de la base de datos abarquen varios hosts en un sistema distribuido, pero para ello se requiere una mayor inversión. Las bases de datos relacionales presentan tamaños máximos en atención al número y al tamaño de los archivos, lo que impone límites máximos de escalabilidad.	DynamoDB se ha diseñado para escalar horizontalmente por medio de clústeres de hardware distribuidos. Este diseño permite obtener un rendimiento mejorado sin aumentar la latencia. Los clientes especifican sus requisitos de rendimiento y DynamoDB asigna recursos suficientes para satisfacer estos requisitos. No hay límite en cuanto al número de elementos por tabla, ni al tamaño total de dicha tabla.

Diferencias al acceder a una base de datos relacional (SQL) y DynamoDB

Antes de que su aplicación pueda acceder a una base de datos, debe autenticarse para garantizar que la aplicación pueda usar la base de datos. Debe autorizarse para que la aplicación pueda realizar solo las acciones para las que tiene permisos.

En el siguiente diagrama se muestra la interacción de un cliente con una base de datos relacional y con Amazon DynamoDB.





En la tabla siguiente se muestra más información acerca de las tareas de interacción del cliente.

Característica	Sistema de base de datos relacional (RDBMS)	Amazon DynamoDB
Herramientas para obtener acceso a la base de datos	La mayoría de las bases de datos relacionales ofrecen una interfaz de línea de comandos	En la mayoría de los casos, el desarrollador debe escribir el código de aplicació

Característica	Sistema de base de datos relacional (RDBMS)	Amazon DynamoDB
	(CLI), que permite especific ar instrucciones SQL ad-hoc y ver los resultados de forma inmediata.	n. También puede usar Consola de administración de AWS, AWS Command Line Interface (AWS CLI) o NoSQL WorkBench para enviar solicitudes ad-hoc a DynamoDB y ver los resultado s. PartiQL, un lenguaje de consulta compatible con SQL, le permite seleccionar, insertar, actualizar y eliminar datos en DynamoDB.
Conexión a la base de datos	Un programa de aplicació n establece y mantiene una conexión de red con la base de datos. Cuando la aplicación finaliza, termina la conexión.	DynamoDB es un servicio web y las interacciones con él son sin estado. Las aplicaciones no tienen que mantener conexiones de red persistentes. En lugar de ello, la interacción con DynamoDB se produce mediante solicitud es y respuestas HTTP(S).

Característica	Sistema de base de datos relacional (RDBMS)	Amazon DynamoDB
Autenticación	Una aplicación no puede conectarse a la base de datos hasta que se ha autentica do. El RDBMS puede llevar a cabo la autenticación por sí mismo, o bien delegar esta tarea en el sistema operativo host o en un servicio de directorio.	Cada solicitud a DynamoDB debe ir acompañada de una firma criptográfica, que autentica esa solicitud concreta. Los SDK de AWS proporcionan toda la lógica necesaria para crear firmas y firmar solicitudes. Para obtener más información, consulte Firma de solicitud es de la API de AWS en la Referencia general de AWS.
Autorización	Las aplicaciones solo pueden llevar a cabo aquellas acciones para las cuales tienen permiso. Los administr adores de bases de datos o los propietarios de las aplicaciones pueden utilizar las instrucciones `GRANT` y `REVOKE` de SQL para controlar el acceso a los objetos de base de datos (tales como las tablas), los datos (como las filas de una tabla) o la capacidad de emitir determinadas instrucciones de SQL.	En DynamoDB, la autorizac ión se administra mediante AWS Identity and Access Management (IAM). Puede escribir una política de IAM que conceda permisos para un recurso de DynamoDB (por ejemplo, una tabla) y, a continuación, permitir a los usuarios y los roles que utilicen esa política. IAM también presenta control de acceso preciso a los elementos de datos individua les contenidos en las tablas de DynamoDB. Para obtener más información, consulte Identity and Access Management en Amazon DynamoDB.

Característica	Sistema de base de datos relacional (RDBMS)	Amazon DynamoDB
Envío de una solicitud	La aplicación emite una instrucción de SQL para todas las operaciones de base de datos que quiere realizar. Al recibir la instrucción de SQL, el RDBMS comprueba su sintaxis, crea un plan para realizar la operación y, a continuación, ejecuta el plan.	La aplicación envía solicitud es HTTP(S) a DynamoDB. Las solicitudes contienen el nombre de la operación de DynamoDB que hay que realizar, junto con los parámetros. DynamoDB ejecuta la solicitud de forma inmediata.
Recepción de una respuesta	El RDBMS devuelve los resultados de la instrucción de SQL. Si se produce un error, el RDBMS devuelve un estado de error y un mensaje.	DynamoDB devuelve una respuesta HTTP(S) que contiene los resultados de la operación. Si se produce un error, DynamoDB devuelve un estado de error de HTTP y uno o varios mensajes.

Diferencias entre una base de datos relacional (SQL) y DynamoDB al crear una tabla

Las tablas son las estructuras de datos fundamentales tanto en las bases de datos relacionales como en Amazon DynamoDB. Un sistema de administración de bases de datos relacionales (RDBMS) requiere que se defina el esquema de la tabla al crearla. En cambio, las tablas de DynamoDB no tienen esquemas por lo tanto, salvo la clave principal, no hay que definir ningún atributo o tipo de datos al crear la tabla.

En la siguiente sección se compara cómo crearía una tabla con SQL y cómo lo haría con DynamoDB.

Temas

- Creación de una tabla con SQL
- Creación de una tabla con DynamoDB

Creación de una tabla con SQL

Con SQL, usaría la instrucción `CREATE TABLE` para crear una tabla, como se muestra en el ejemplo siguiente.

```
CREATE TABLE Music (
    Artist VARCHAR(20) NOT NULL,
    SongTitle VARCHAR(30) NOT NULL,
    AlbumTitle VARCHAR(25),
    Year INT,
    Price FLOAT,
    Genre VARCHAR(10),
    Tags TEXT,
    PRIMARY KEY(Artist, SongTitle)
);
```

La clave principal de esta tabla consta de Artist y SongTitle.

Debe definir la clave principal, así como todas las columnas y tipos de datos de la tabla. Puede usar la instrucción `ALTER TABLE` para cambiar estas definiciones más adelante si fuera preciso.

Muchas implementaciones de SQL permiten definir especificaciones de almacenamiento de la tabla en la propia instrucción `CREATE TABLE`. A menos que se indique otra cosa, la tabla se crea con los ajustes de almacenamiento predeterminados. En un entorno de producción, un administrador de base de datos puede ayudar a determinar los parámetros de almacenamiento óptimos.

Creación de una tabla con DynamoDB

Use la operación `CreateTable` para crear una tabla en modo aprovisionado; para ello, especifique los parámetros que se muestran a continuación:

```
{
    TableName : "Music",
    KeySchema: [
        {
            AttributeName: "Artist",
            KeyType: "HASH" //Partition key
        },
        {
            AttributeName: "SongTitle",
            KeyType: "RANGE" //Sort key
        }
```

```
    ],
    AttributeDefinitions: [
        {
            AttributeName: "Artist",
            AttributeType: "S"
        },
        {
            AttributeName: "SongTitle",
            AttributeType: "S"
        }
    ],
    ProvisionedThroughput: {       // Only specified if using provisioned mode
        ReadCapacityUnits: 1,
        WriteCapacityUnits: 1
    }
}
```

La clave principal de esta tabla consta de Artist (clave de partición) y SongTitle (clave de ordenación).

Debe proporcionar los siguientes parámetros a `CreateTable`:

- `TableName`: nombre de la tabla.
- `KeySchema`: atributos que se utilizan para la clave principal. Para obtener más información, consulte Tablas, elementos y atributos y Clave principal.
- `AttributeDefinitions`: tipos de datos de los atributos del esquema de claves.
- `ProvisionedThroughput (for provisioned tables)`: número de lecturas y escrituras por segundo que se requieren para esta tabla. DynamoDB reserva recursos de almacenamiento y del sistema suficientes para cumplir en todo momento los requisitos de rendimiento. Puede usar la operación `UpdateTable` para cambiar estos valores más adelante si fuera preciso. No es necesario especificar los requisitos de almacenamiento de una tabla, porque DynamoDB se encarga de administrar todas las asignaciones del almacenamiento.

Diferencias entre obtener información de tablas de una base de datos relacional (SQL) y DynamoDB

Puede comprobar que una tabla se ha creado de acuerdo con sus especificaciones. En una base de datos relacional, se muestra el esquema completo de la tabla. Las tablas de Amazon DynamoDB no tienen esquema, por lo que solo se muestran los atributos de clave principal.

Temas

- Obtención de información sobre una tabla con SQL
- Obtención de información sobre una tabla en DynamoDB

Obtención de información sobre una tabla con SQL

La mayoría de los sistemas de administración de bases de datos relacionales (RDBMS) permiten describir una estructura de tabla; a saber, columnas, tipos de datos, definición de clave principal, etc. No existe una manera estándar de realizar esta tarea en SQL. Sin embargo, en muchos sistemas de base de datos se proporciona un comando `DESCRIBE`. A continuación se muestra un ejemplo de MySQL.

```
DESCRIBE Music;
```

Este código devuelve la estructura de la tabla, con todos los nombres de columnas, los tipos de datos y los tamaños.

```
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| Artist     | varchar(20) | NO   | PRI | NULL    |       |
| SongTitle  | varchar(30) | NO   | PRI | NULL    |       |
| AlbumTitle | varchar(25) | YES  |     | NULL    |       |
| Year       | int(11)     | YES  |     | NULL    |       |
| Price      | float       | YES  |     | NULL    |       |
| Genre      | varchar(10) | YES  |     | NULL    |       |
| Tags       | text        | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

La clave principal de esta tabla consta de Artist y SongTitle.

Obtención de información sobre una tabla en DynamoDB

DynamoDB posee una operación `DescribeTable`, que es parecida. El único parámetro es el nombre de la tabla.

```
{
    TableName : "Music"
}
```

La respuesta de `DescribeTable` tiene el siguiente aspecto.

```
{
  "Table": {
    "AttributeDefinitions": [
      {
        "AttributeName": "Artist",
        "AttributeType": "S"
      },
      {
        "AttributeName": "SongTitle",
        "AttributeType": "S"
      }
    ],
    "TableName": "Music",
    "KeySchema": [
      {
        "AttributeName": "Artist",
        "KeyType": "HASH"  //Partition key
      },
      {
        "AttributeName": "SongTitle",
        "KeyType": "RANGE"  //Sort key
      }
    ],

    ...
```

`DescribeTable` también proporciona información acerca de los índices de la tabla, los ajustes de rendimiento aprovisionado, el recuento de elementos aproximado y otros metadatos.

Diferencias entre una base de datos relacional (SQL) y DynamoDB al escribir datos en una tabla

Las tablas de las bases de datos relacionales contienen filas de datos. Las filas constan de columnas. Las tablas de Amazon DynamoDB contienen elementos. Los elementos constan de atributos.

En esta sección se describe cómo escribir una fila (o un elemento) en una tabla.

Temas

- Escritura de datos en una tabla con SQL

- Escritura de datos en una tabla de DynamoDB

Escritura de datos en una tabla con SQL

Una tabla de una base de datos relacional es una estructura de datos bidimensional formada por filas y columnas. Algunos sistemas de administración de bases de datos también ofrecen compatibilidad con datos semiestructurados, normalmente con los tipos de datos de JSON o XML nativos. Sin embargo, los detalles de implementación varían según el proveedor.

En SQL, se utilizaría la instrucción `INSERT` para agregar una fila a una tabla.

```
INSERT INTO Music
    (Artist, SongTitle, AlbumTitle,
    Year, Price, Genre,
    Tags)
VALUES(
    'No One You Know', 'Call Me Today', 'Somewhat Famous',
    2015, 2.14, 'Country',
    '{"Composers": ["Smith", "Jones", "Davis"],"LengthInSeconds": 214}'
);
```

La clave principal de esta tabla consta de Artist y SongTitle. Debe especificar los valores de estas columnas.

> **Note**
>
> En este ejemplo se utiliza la columna Tags para almacenar datos semiestructurados relativos a las canciones de la tabla Music. La columna Tags se define como el tipo TEXT, que permite almacenar hasta 65,535 caracteres en MySQL.

Escritura de datos en una tabla de DynamoDB

En Amazon DynamoDB, puede utilizar la API de DynamoDB o PartiQL (un lenguaje de consulta compatible con SQL) para agregar un elemento a una tabla.

DynamoDB API

Con la API de DynamoDB, se utiliza la operación `PutItem` para agregar un elemento a una tabla.

```
{
    TableName: "Music",
    Item: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today",
        "AlbumTitle":"Somewhat Famous",
        "Year": 2015,
        "Price": 2.14,
        "Genre": "Country",
        "Tags": {
            "Composers": [
                  "Smith",
                  "Jones",
                  "Davis"
            ],
            "LengthInSeconds": 214
        }
    }
}
```

La clave principal de esta tabla consta de Artist y SongTitle. Debe especificar los valores de estos atributos.

A continuación se indican algunos aspectos clave que es preciso tener en cuenta sobre este ejemplo de `PutItem`:

- DynamoDB proporciona compatibilidad nativa con documentos mediante JSON. Por ello, DynamoDB resulta idóneo para almacenar datos semiestructurados, como las etiquetas. Además, puede recuperar y manipular los datos contenidos en los documentos JSON.
- La tabla Music no tiene atributos predefinidos aparte de la clave principal (Artist y SongTitle).
- La mayoría de las bases de datos SQL están orientadas a transacciones. Cuando se emite una instrucción `INSERT`, las modificaciones de los datos no son permanentes hasta que se emite una instrucción `COMMIT`. Con Amazon DynamoDB, los efectos de una operación `PutItem` son permanentes cuando DynamoDB responde con un código de estado HTTP 200 (`OK`).

A continuación, se muestran algunos otros ejemplos de `PutItem`.

```
{
    TableName: "Music",
    Item: {
```

```
        "Artist": "No One You Know",
        "SongTitle": "My Dog Spot",
        "AlbumTitle":"Hey Now",
        "Price": 1.98,
        "Genre": "Country",
        "CriticRating": 8.4
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "No One You Know",
        "SongTitle": "Somewhere Down The Road",
        "AlbumTitle":"Somewhat Famous",
        "Genre": "Country",
        "CriticRating": 8.4,
        "Year": 1984
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "The Acme Band",
        "SongTitle": "Still In Love",
        "AlbumTitle":"The Buck Starts Here",
        "Price": 2.47,
        "Genre": "Rock",
        "PromotionInfo": {
            "RadioStationsPlaying":[
                 "KHCR", "KBQX", "WTNR", "WJJH"
            ],
            "TourDates": {
                "Seattle": "20150625",
                "Cleveland": "20150630"
            },
            "Rotation": "Heavy"
        }
    }
}
```

```
{
    TableName: "Music",
    Item: {
        "Artist": "The Acme Band",
        "SongTitle": "Look Out, World",
        "AlbumTitle":"The Buck Starts Here",
        "Price": 0.99,
        "Genre": "Rock"
    }
}
```

> **Note**
>
> Además de `PutItem`, DynamoDB admite la operación `BatchWriteItem` para escribir varios elementos a la vez.

PartiQL for DynamoDB

Con PartiQL, se utiliza la operación `ExecuteStatement` para agregar un elemento a una tabla, mediante la instrucción `Insert` de PartiQL.

```
INSERT into Music value {
    'Artist': 'No One You Know',
    'SongTitle': 'Call Me Today',
    'AlbumTitle': 'Somewhat Famous',
    'Year' : '2015',
    'Genre' : 'Acme'
}
```

La clave principal de esta tabla consta de Artist y SongTitle. Debe especificar los valores de estos atributos.

> **Note**
>
> Para obtener ejemplos de código mediante `Insert` y `ExecuteStatement`, consulte Instrucciones de inserción de PartiQL para DynamoDB.

Diferencias entre una base de datos relacional (SQL) y DynamoDB al leer datos de una tabla

Con SQL, se utiliza la instrucción `SELECT` para recuperar una o varias filas de una tabla. Se utiliza la cláusula `WHERE` para determinar qué datos se devuelven.

Es diferente de utilizar Amazon DynamoDB, que proporciona las siguientes operaciones para la lectura de datos:

- `ExecuteStatement` recupera uno o varios elementos de una tabla. `BatchExecuteStatement` recupera varios elementos de tablas diferentes en una sola operación. Ambas operaciones utilizan PartiQL, un lenguaje de consulta compatible con SQL.
- `GetItem`: recupera un solo elemento de una tabla. Se trata de la forma más eficiente de leer un único elemento, ya que proporciona acceso directo a la ubicación física del elemento. (DynamoDB también proporciona la operación `BatchGetItem`, que permite llevar a cabo hasta 100 llamadas a `GetItem` en una sola operación).
- `Query`: recupera todos los elementos que tienen una clave de partición determinada. Dentro del conjunto de esos elementos, puede aplicar una condición a la clave de ordenación y recuperar únicamente un subconjunto de los datos. `Query` proporciona un acceso rápido y eficiente a las particiones en las que se almacenan los datos. (Para obtener más información, consulte). Particiones y distribución de datos en DynamoDB.)
- `Scan`: recupera todos los elementos de la tabla especificada. (Esta operación no debe utilizarse con grandes tablas, ya que puede consumir gran cantidad de recursos del sistema).

> **Note**
>
> En una base de datos relacional, puede usar la instrucción `SELECT` para unir los datos de varias tablas y devolver los resultados. Las uniones son fundamentales para el modelo relacional. Con el fin de garantizar que las uniones se ejecuten de forma eficiente, es preciso ajustar continuamente el desempeño de la base de datos y de sus aplicaciones. DynamoDB es una base de datos NoSQL no relacional que no admite uniones de tablas. En lugar de ello, las aplicaciones leen los datos de una tabla cada vez.

En las siguientes secciones se describen varios casos de uso de la lectura de datos y se explica cómo realizar estas tareas con una base de datos relacional y con DynamoDB.

Temas

- Diferencias al leer un elemento mediante la clave principal
- Diferencias al consultar una tabla
- Diferencias al analizar una tabla

Diferencias al leer un elemento mediante la clave principal

Un patrón de acceso habitual de acceso a las bases de datos consiste en leer un único elemento de una tabla. Es preciso especificar la clave principal del elemento que se desea.

Temas

- Lectura de un elemento usando su clave principal con SQL
- Lectura de un elemento usando su clave principal en DynamoDB

Lectura de un elemento usando su clave principal con SQL

En SQL, se utilizaría la instrucción `SELECT` para recuperar datos de una tabla. Puede solicitar una o varias columnas en el resultado (o todas ellas, si se utiliza el operador `*`). La cláusula `WHERE` determina qué filas se devolverán.

A continuación se muestra una instrucción `SELECT` que recupera una sola fila de la tabla Music. La cláusula `WHERE` especifica los valores de la clave principal.

```
SELECT *
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Puede modificar esta consulta para recuperar tan solo un subconjunto de las columnas:

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Observe que la clave principal de esta tabla consta de Artist y SongTitle.

Lectura de un elemento usando su clave principal en DynamoDB

En Amazon DynamoDB, puede utilizar la API de DynamoDB o PartiQL (un lenguaje de consulta compatible con SQL) para leer un elemento de una tabla.

DynamoDB API

Con la API de DynamoDB, se utiliza la operación `PutItem` para agregar un elemento a una tabla.

DynamoDB proporciona la operación `GetItem` para recuperar un elemento por su clave principal. `GetItem` es muy eficiente ya que proporciona acceso directo a la ubicación física del elemento. (Para obtener más información, consulte). Particiones y distribución de datos en DynamoDB.)

De forma predeterminada, `GetItem` devuelve el elemento completo con todos sus atributos.

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    }
}
```

Puede agregar un parámetro `ProjectionExpression` para que solo se devuelvan algunos de los atributos.

```
{
    TableName: "Music",
    Key: {
        "Artist": "No One You Know",
        "SongTitle": "Call Me Today"
    },
    "ProjectionExpression": "AlbumTitle, Year, Price"
}
```

Observe que la clave principal de esta tabla consta de Artist y SongTitle.

La operación `GetItem` de DynamoDB es muy eficiente. Utiliza los valores de clave principal para determinar la ubicación de almacenamiento exacta del elemento en cuestión y lo recupera directamente desde ella. La eficiencia de la instrucción `SELECT` de SQL es parecida para recuperar elementos por sus valores de clave principal.

La instrucción `SELECT` de SQL admite muchos tipos de consultas y exámenes de tablas. DynamoDB proporciona una funcionalidad semejante con las operaciones `Query` y `Scan`, que se describen en Diferencias al consultar una tabla y Diferencias al analizar una tabla.

La instrucción `SELECT` de SQL puede llevar a cabo uniones de tablas para permitirle recuperar datos de varias tablas al mismo tiempo. Las uniones son más eficaces cuando las tablas de base de datos están normalizadas y las relaciones entre las tablas están claras. Sin embargo, si se unen demasiadas tablas en una misma instrucción `SELECT`, el rendimiento de la aplicación podría verse afectado. Puede solucionar este problema utilizando la replicación de bases de datos, las vistas materializadas o las reescrituras de consultas.

DynamoDB es una base de datos no relacional y no admite uniones de tablas. Si va a migrar una aplicación existente a partir de una base de datos relacional a DynamoDB, debe desnormalizar el modelo de datos para eliminar la necesidad de uniones.

PartiQL for DynamoDB

Con PartiQL, se utiliza la operación `ExecuteStatement` para leer un elemento de una tabla, mediante la instrucción `Select` de PartiQL.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today'
```

Observe que la clave principal de esta tabla consta de Artist y SongTitle.

> **Note**
>
> La instrucción select de PartiQL también se puede utilizar para consultar o analizar una tabla de DynamoDB

Para obtener ejemplos de código mediante `Select` y `ExecuteStatement`, consulte Instrucciones de selección de PartiQL para DynamoDB.

Diferencias al consultar una tabla

Otro patrón de acceso habitual consiste en leer varios elementos de una tabla, según los criterios de consulta.

Temas

- Consulta de una tabla con SQL
- Consulta de una tabla en DynamoDB

Consulta de una tabla con SQL

Al utilizar SQL, la instrucción `SELECT` permite realizar consultas por columnas de clave, columnas que no son de clave o cualquier combinación de ellas. La cláusula `WHERE` determina qué filas se devuelven, como se muestra en los ejemplos siguientes.

```
/* Return a single song, by primary key */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle = 'Call Me Today';
```

```
/* Return all of the songs by an artist */

SELECT * FROM Music
WHERE Artist='No One You Know';
```

```
/* Return all of the songs by an artist, matching first part of title */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle LIKE 'Call%';
```

```
/* Return all of the songs by an artist, with a particular word in the title...
...but only if the price is less than 1.00 */

SELECT * FROM Music
WHERE Artist='No One You Know' AND SongTitle LIKE '%Today%'
AND Price < 1.00;
```

Observe que la clave principal de esta tabla consta de Artist y SongTitle.

Consulta de una tabla en DynamoDB

En Amazon DynamoDB, puede utilizar la API de DynamoDB o PartiQL (un lenguaje de consulta compatible con SQL) para consultar un elemento de una tabla.

DynamoDB API

Con Amazon DynamoDB, puede utilizar la operación `Query` para recuperar datos de manera parecida. La operación `Query` proporciona un acceso rápido y eficiente a las ubicaciones físicas en las que se almacenan los datos. Para obtener más información, consulte Particiones y distribución de datos en DynamoDB.

Se puede utilizar `Query` con cualquier tabla o índice secundario. Debe especificar una condición de igualdad para el valor de la clave de partición y, si lo desea, puede proporcionar otra condición para el atributo de clave de clasificación si está definido.

El parámetro `KeyConditionExpression` especifica los valores de clave que se desea consultar. Puede utilizar una expresión `FilterExpression` opcional para eliminar algunos elementos de los resultados antes de que se devuelvan.

En DynamoDB, debe usar `ExpressionAttributeValues` como marcador de posición en los parámetros de las expresiones (tales como `KeyConditionExpression` y `FilterExpression`). Esto es análogo al uso de las variables de vínculo de las bases de datos relacionales, que se sustituyen en la instrucción `SELECT` por los valores reales en tiempo de ejecución.

Observe que la clave principal de esta tabla consta de Artist y SongTitle.

A continuación, se muestran algunos ejemplos de `Query` de DynamoDB.

```
// Return a single song, by primary key

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and SongTitle = :t",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Call Me Today"
    }
}
```

```
// Return all of the songs by an artist

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a",
```

```
    ExpressionAttributeValues: {
        ":a": "No One You Know"
    }
}
```

```
// Return all of the songs by an artist, matching first part of title

{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and begins_with(SongTitle, :t)",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Call"
    }
}
```

PartiQL for DynamoDB

Con PartiQL, puede hacer una consulta mediante la operación `ExecuteStatement` y la instrucción `Select` sobre la clave de partición.

```
SELECT AlbumTitle, Year, Price
FROM Music
WHERE Artist='No One You Know'
```

El uso de la instrucción `SELECT` de esta manera devuelve todas las canciones asociadas a este `Artist` en particular.

Para obtener ejemplos de código mediante `Select` y `ExecuteStatement`, consulte Instrucciones de selección de PartiQL para DynamoDB.

Diferencias al analizar una tabla

En SQL, una instrucción `SELECT` sin cláusula `WHERE` devuelve todas las filas de la tabla. En Amazon DynamoDB, la operación `Scan` hace lo mismo. En ambos casos, puede recuperar todos los elementos o solo algunos de ellos.

Tanto si va a usar una base de datos SQL o NoSQL, los exámenes deben utilizarse con moderación, ya que puede consumir gran cantidad de recursos del sistema. En ocasiones, un examen es lo apropiado (por ejemplo, para examinar una tabla pequeña) o algo inevitable (por ejemplo, para

realizar una exportación masiva de datos). Sin embargo, por norma general, debe diseñar las aplicaciones de tal forma que se evite la realización de exámenes. Para obtener más información, consulte Consulta de tablas en DynamoDB.

> **Note**
>
> Realizar una exportación masiva también crea al menos 1 archivo por partición. Todos los elementos de cada archivo provienen del espacio de claves hash de esa partición en particular.

Temas

- Análisis de una tabla con SQL
- Análisis de una tabla en DynamoDB

Análisis de una tabla con SQL

Al utilizar SQL, puede examinar una tabla y recuperar todos sus datos utilizando una instrucción `SELECT` sin especificar una cláusula `WHERE`. Puede solicitar una o más columnas en el resultado. O bien puede solicitar todas ellas si utiliza el carácter comodín (*).

En los siguientes ejemplos se utiliza una instrucción `SELECT`.

```
/* Return all of the data in the table */
SELECT * FROM Music;
```

```
/* Return all of the values for Artist and Title */
SELECT Artist, Title FROM Music;
```

Análisis de una tabla en DynamoDB

En Amazon DynamoDB, puede utilizar la API de DynamoDB o PartiQL (un lenguaje de consulta compatible con SQL) para analizar una tabla.

DynamoDB API

Con la API de DynamoDB, el usuario utiliza la operación `Scan` para devolver uno o más elementos y atributos de elementos mediante el acceso a todos los elementos de una tabla o un índice secundario.

```
// Return all of the data in the table
{
    TableName:  "Music"
}
```

```
// Return all of the values for Artist and Title
{
    TableName:  "Music",
    ProjectionExpression: "Artist, Title"
}
```

La operación `Scan` proporciona además un parámetro `FilterExpression`, que se puede usar para descartar elementos que no desee que aparezcan en los resultados. La expresión `FilterExpression` se aplica después de que analizar la tabla completa, pero antes de que se devuelvan los resultados. (Esto no se recomienda en tablas de gran tamaño, porque se le cobrará la operación `Scan` completa aunque solamente se devuelvan algunos elementos coincidentes).

PartiQL for DynamoDB

Con PartiQL, se hace un análisis mediante la operación `ExecuteStatement` para devolver todo el contenido de una tabla mediante la instrucción `Select`.

```
SELECT AlbumTitle, Year, Price
FROM Music
```

Tenga en cuenta que esta instrucción devolverá todos los elementos de la tabla Música.

Para obtener ejemplos de código mediante `Select` y `ExecuteStatement`, consulte Instrucciones de selección de PartiQL para DynamoDB.

Diferencias entre una base de datos relacional (SQL) y DynamoDB al administrar índices

Los índices permiten obtener acceso a patrones de consulta alternativos y agilizar las consultas. En esta sección se compara y contrasta la creación y el uso de índices entre SQL y Amazon DynamoDB.

Tanto si está utilizando una base de datos relacional o DynamoDB, la creación de índices debe realizarse con cautela. Cada vez que se produce una escritura en una tabla, es preciso actualizar

todos los índices de esa tabla. En un entorno con uso intensivo de escrituras y grandes tablas, esto puede consumir gran cantidad de recursos del sistema. Esto no es motivo de preocupación en entornos de solo lectura o principalmente de lectura. Sin embargo, hay que asegurarse de que la aplicación utilice los índices realmente, de tal forma que estos no se limiten a consumir espacio.

Temas

- Diferencias entre una base de datos relacional (SQL) y DynamoDB al crear un índice
- Diferencias entre una base de datos relacional (SQL) y DynamoDB al consultar y analizar un índice

Diferencias entre una base de datos relacional (SQL) y DynamoDB al crear un índice

Compare la instrucción `CREATE INDEX` en SQL con la operación `UpdateTable` en Amazon DynamoDB.

Temas

- Creación de un índice con SQL
- Creación de un índice en DynamoDB

Creación de un índice con SQL

En las bases de datos relacionales, los índices son estructuras de datos que permiten realizar consultas rápidas en diferentes columnas de una tabla. Puede usar la instrucción `CREATE INDEX` de SQL para agregar un índice a una tabla existente y especificar las columnas que se van a indexar. Una vez que se ha creado el índice, puede consultar los datos de la tabla como de costumbre, si bien ahora la base de datos podrá utilizar el índice para encontrar rápidamente las filas especificadas de la tabla, en lugar de tener que examinar la tabla completa.

Después de crear un índice, la base de datos lo mantiene automáticamente. Cada vez que se modifica algún dato de la tabla, el índice se modifica automáticamente para reflejar los cambios de la tabla.

En MySQL, se crearía un índice como se indica a continuación.

```
CREATE INDEX GenreAndPriceIndex
ON Music (genre, price);
```

Creación de un índice en DynamoDB

En DynamoDB puede crear y usar un índice secundario con fines semejantes.

Los índices de DynamoDB son diferentes de sus homólogos relacionales. Al crear un índice secundario, hay que especificar sus atributos de clave: una clave de partición y una clave de ordenación. Una vez creado el índice secundario, puede utilizar las operaciones `Query` o `Scan` para consultarlo o examinarlo, respectivamente, como si se tratase de una tabla. DynamoDB no tiene un optimizador de consultas, de modo que el índice secundario solamente se utiliza cuando se utilizan las operaciones `Query` o `Scan` con él.

DynamoDB admite dos tipos distintos de índices:

- Los índices secundarios globales: la clave principal del índice puede constar de dos atributos cualesquiera de la tabla a la que pertenece.
- Los índices secundarios locales: la clave de partición del índice debe ser igual que la clave de partición de la tabla a la que pertenece. Sin embargo, la clave de ordenación puede ser cualquier otro atributo.

DynamoDB se asegura de que los datos de un índice secundario presenten consistencia final con la tabla a la que pertenece. Puede solicitar operaciones `Query` o `Scan` de consistencia alta en una tabla o un índice. No obstante, los índices secundarios globales solo admiten la consistencia final.

Puede agregar un índice secundario global a una tabla existente; para ello, utilice la operación `UpdateTable` y especifique `GlobalSecondaryIndexUpdates`.

```
{
    TableName: "Music",
    AttributeDefinitions:[
        {AttributeName: "Genre", AttributeType: "S"},
        {AttributeName: "Price", AttributeType: "N"}
    ],
    GlobalSecondaryIndexUpdates: [
        {
            Create: {
                IndexName: "GenreAndPriceIndex",
                KeySchema: [
                    {AttributeName: "Genre", KeyType: "HASH"}, //Partition key
                    {AttributeName: "Price", KeyType: "RANGE"}, //Sort key
                ],
```

```
                Projection: {
                    "ProjectionType": "ALL"
                },
                ProvisionedThroughput: {                                    // Only
 specified if using provisioned mode
                    "ReadCapacityUnits": 1,"WriteCapacityUnits": 1
                }
            }
        }
    ]
}
```

Debe proporcionar los siguientes parámetros a `UpdateTable`:

- `TableName`: tabla a la que se asociará el índice.
- `AttributeDefinitions`: tipos de datos de los atributos de esquema de claves del índice.
- `GlobalSecondaryIndexUpdates`: detalles sobre el índice que se desea crear.
  - `IndexName`: nombre del índice.
  - `KeySchema`: atributos utilizados para la clave principal del índice.
  - `Projection`: atributos de la tabla que se copian en el índice. En este caso, `ALL` significa que se copian todos los atributos.
  - `ProvisionedThroughput (for provisioned tables)`: número de lecturas y escrituras por segundo que se requieren para este índice. Este valor es independiente de los ajustes de rendimiento aprovisionado de la tabla.

Parte de esta operación implica reponer datos de la tabla en el nuevo índice. Durante la reposición, la tabla permanece disponible. Sin embargo, el índice no está preparado hasta que el atributo `Backfilling` cambia de true a false. Puede utilizar la operación `DescribeTable` para ver este atributo.

Diferencias entre una base de datos relacional (SQL) y DynamoDB al consultar y analizar un índice

Compare la realización de consultas y análisis de un índice mediante la instrucción SELECT de SQL con las operaciones `Query` y `Scan` en Amazon DynamoDB.

Temas

- Consulta y análisis de un índice con SQL

- Consulta y análisis de un índice en DynamoDB

Consulta y análisis de un índice con SQL

En una base de datos relacional, no se trabajar directamente con los índices. En lugar de ello, se consultan las tablas mediante instrucciones `SELECT` y el optimizador de consultas utiliza los índices, si los hay.

Un optimizador de consultas es un componente del sistema de administración de bases de datos relacionales (RDBMS) que se encarga de evaluar los índices disponibles y determinar si se pueden utilizar para agilizar consultas. Si los índices pueden utilizarse para agilizar una consulta, el RDBMS obtiene acceso al índice en primer lugar y, a continuación, lo utiliza para localizar los datos en la tabla.

A continuación se muestran algunas instrucciones SQL en las que se puede utilizar GenreAndPriceIndex para mejorar el rendimiento. Damos por hecho que la tabla Music contiene suficientes datos para que el optimizador de consultas decida utilizar el índice en lugar de examinar la tabla completa.

```
/* All of the rock songs */

SELECT * FROM Music
WHERE Genre = 'Rock';
```

```
/* All of the cheap country songs */

SELECT Artist, SongTitle, Price FROM Music
WHERE Genre = 'Country' AND Price < 0.50;
```

Consulta y análisis de un índice en DynamoDB

En DynamoDB, se llevan a cabo las operaciones `Query` y `Scan` que consultan el índice directamente, de la misma manera en la que se haría en una tabla. Puede utilizar la API de DynamoDB o PartiQL (un lenguaje de consulta compatible con SQL) para consultar o analizar el índice. Debe especificar tanto `TableName` como `IndexName`.

A continuación se muestran algunas consultas sobre GenreAndPriceIndex en DynamoDB. El esquema de claves de este índice consta de Genre y Price.

DynamoDB API

```
// All of the rock songs

{
    TableName: "Music",
    IndexName: "GenreAndPriceIndex",
    KeyConditionExpression: "Genre = :genre",
    ExpressionAttributeValues: {
        ":genre": "Rock"
    },
};
```

En este ejemplo se utiliza una expresión `ProjectionExpression` para indicar que solamente deseamos que aparezcan en los resultados algunos de los atributos y no todos ellos.

```
// All of the cheap country songs

{
    TableName: "Music",
    IndexName: "GenreAndPriceIndex",
    KeyConditionExpression: "Genre = :genre and Price < :price",
    ExpressionAttributeValues: {
        ":genre": "Country",
        ":price": 0.50
    },
    ProjectionExpression: "Artist, SongTitle, Price"
};
```

A continuación se muestra un examen de GenreAndPriceIndex.

```
// Return all of the data in the index

{
    TableName:  "Music",
    IndexName: "GenreAndPriceIndex"
}
```

PartiQL for DynamoDB

Con PartiQL, se utiliza la instrucción `Select` de PartiQL para hacer consultas y análisis en el índice.

```
// All of the rock songs

SELECT *
FROM Music.GenreAndPriceIndex
WHERE Genre = 'Rock'
```

```
// All of the cheap country songs

SELECT *
FROM Music.GenreAndPriceIndex
WHERE Genre = 'Rock' AND Price < 0.50
```

A continuación se muestra un examen de GenreAndPriceIndex.

```
// Return all of the data in the index

SELECT *
FROM Music.GenreAndPriceIndex
```

> **Note**
>
> Para obtener ejemplos de código en los que se utiliza `Select`, consulte Instrucciones de selección de PartiQL para DynamoDB.

Diferencias entre una base de datos relacional (SQL) y DynamoDB al modificar datos de una tabla

El lenguaje SQL proporciona la instrucción `UPDATE` para modificar datos. Amazon DynamoDB utiliza la operación `UpdateItem` para llevar a cabo tareas semejantes.

Temas

- Modificación de los datos de una tabla con SQL
- Modificación de los datos de una tabla en DynamoDB

Modificación de los datos de una tabla con SQL

En SQL, se utilizaría la instrucción `UPDATE` para modificar una o varias filas. La cláusula `SET` especifica los nuevos valores de una o varias columnas y la cláusula `WHERE` determina qué filas se modifican. A continuación se muestra un ejemplo.

```
UPDATE Music
SET RecordLabel = 'Global Records'
WHERE Artist = 'No One You Know' AND SongTitle = 'Call Me Today';
```

Si no hay ninguna fila que coincida con la cláusula `WHERE`, la instrucción `UPDATE` no surte efecto.

Modificación de los datos de una tabla en DynamoDB

En DynamoDB, puede utilizar la API clásica o PartiQL (un lenguaje de consulta compatible con SQL) para modificar un solo elemento. Si desea modificar varios elementos, debe utilizar varias operaciones.

DynamoDB API

Con la API de DynamoDB, se utiliza la operación `UpdateItem` para modificar un solo elemento.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET RecordLabel = :label",
    ExpressionAttributeValues: {
        ":label": "Global Records"
    }
}
```

Debe especificar los atributos `Key` del elemento que va a modificar y una expresión `UpdateExpression` para especificar los valores de los atributos. `UpdateItem` se comporta como una operación “upsert”. Si el elemento está presente en tabla, se actualiza y, si no está presente, se agrega (inserta).

`UpdateItem` admite las escrituras condicionales, en las que la operación únicamente se lleva a cabo correctamente si una expresión `ConditionExpression` determinada se evalúa en true.

Por ejemplo, la operación `UpdateItem` siguiente no lleva a cabo la actualización a no ser que el precio de la canción sea mayor o igual que 2,00.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET RecordLabel = :label",
    ConditionExpression: "Price >= :p",
    ExpressionAttributeValues: {
        ":label": "Global Records",
        ":p": 2.00
    }
}
```

`UpdateItem` también admite los contadores atómicos, que son atributos del tipo `Number` que se pueden incrementar o reducir. Los contadores atómicos se parecen en muchos aspectos a los generadores de secuencia, las columnas de identidad o los campos de incremento automático de las bases de datos SQL.

A continuación se muestra un ejemplo de operación `UpdateItem` utilizada para inicializar un nuevo atributo (Plays) que permite realizar el seguimiento del número de veces que se ha reproducido una canción.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET Plays = :val",
    ExpressionAttributeValues: {
        ":val": 0
    },
    ReturnValues: "UPDATED_NEW"
}
```

El parámetro `ReturnValues` se establece en `UPDATED_NEW`, que devuelve los nuevos valores de todos los atributos que se han actualizado. En este caso, devuelve 0 (cero).

Cada vez que alguien reproduce esta canción, podemos usar la operación `UpdateItem` para incrementar Plays en una unidad.

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET Plays = Plays + :incr",
    ExpressionAttributeValues: {
        ":incr": 1
    },
    ReturnValues: "UPDATED_NEW"
}
```

PartiQL for DynamoDB

Con PartiQL, se utiliza la operación `ExecuteStatement` para modificar un elemento de una tabla, mediante la instrucción `Update` de PartiQL.

La clave principal de esta tabla consta de Artist y SongTitle. Debe especificar los valores de estos atributos.

```
UPDATE Music
SET RecordLabel ='Global Records'
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

También puede modificar varios campos a la vez, como en el ejemplo siguiente.

```
UPDATE Music
SET RecordLabel = 'Global Records'
SET AwardsWon = 10
WHERE Artist ='No One You Know' AND SongTitle='Call Me Today'
```

`Update` también admite los contadores atómicos, que son atributos del tipo `Number` que se pueden incrementar o reducir. Los contadores atómicos se parecen en muchos aspectos a los generadores de secuencia, las columnas de identidad o los campos de incremento automático de las bases de datos SQL.

A continuación, se muestra un ejemplo de instrucción `Update` utilizada para inicializar un nuevo atributo (Plays [Reproducciones]) que permite hacer el seguimiento del número de veces que se ha reproducido una canción.

```
UPDATE Music
SET Plays = 0
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

Cada vez que alguien reproduce esta canción, podemos usar la instrucción `Update` para aumentar Plays (Reproducciones) en una unidad.

```
UPDATE Music
SET Plays = Plays + 1
WHERE Artist='No One You Know' AND SongTitle='Call Me Today'
```

> **Note**
>
> Para obtener ejemplos de código mediante `Update` y `ExecuteStatement`, consulte Instrucciones de actualización de PartiQL para DynamoDB.

Diferencias entre una base de datos relacional (SQL) y DynamoDB al eliminar datos de una tabla

En SQL, se utiliza la instrucción `DELETE` para eliminar una o varias filas de una tabla. Amazon DynamoDB utiliza la operación `DeleteItem` para eliminar los elementos de uno en uno.

Temas

- Eliminación de datos de una tabla con SQL
- Eliminación de datos de una tabla en DynamoDB

Eliminación de datos de una tabla con SQL

En SQL, se utiliza la instrucción `DELETE` para eliminar una o varias filas. La cláusula `WHERE` determina qué filas se van a modificar. A continuación se muestra un ejemplo.

```
DELETE FROM Music
```

```
WHERE Artist = 'The Acme Band' AND SongTitle = 'Look Out, World';
```

Puede modificar la cláusula `WHERE` para eliminar varias filas. Por ejemplo, puede eliminar todas las canciones de un determinado intérprete, como se muestra en el siguiente ejemplo.

```
DELETE FROM Music WHERE Artist = 'The Acme Band'
```

Eliminación de datos de una tabla en DynamoDB

En DynamoDB, puede utilizar la API clásica o PartiQL (un lenguaje de consulta compatible con SQL) para eliminar un solo elemento. Si desea modificar varios elementos, debe utilizar varias operaciones.

DynamoDB API

Con la API de DynamoDB, se utiliza la operación `DeleteItem` para eliminar datos de una tabla, de elemento en elemento. Debe especificar los valores de clave principal del elemento.

```
{
    TableName: "Music",
    Key: {
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    }
}
```

> **Note**
>
> Además de `DeleteItem`, Amazon DynamoDB admite la operación `BatchWriteItem` para eliminar varios elementos a la vez.

`DeleteItem` admite las escrituras condicionales, en las que la operación únicamente se lleva a cabo correctamente si una expresión `ConditionExpression` determinada se evalúa en true. Por ejemplo, la operación `DeleteItem` siguiente elimina un elemento únicamente si tiene el atributo RecordLabel.

```
{
    TableName: "Music",
    Key: {
```

```
        Artist: "The Acme Band",
        SongTitle: "Look Out, World"
    },
   ConditionExpression: "attribute_exists(RecordLabel)"
}
```

PartiQL for DynamoDB

Con PartiQL, se utiliza la instrucción `Delete` mediante la operación `ExecuteStatement` para eliminar datos de una tabla, de elemento en elemento. Debe especificar los valores de clave principal del elemento.

La clave principal de esta tabla consta de Artist y SongTitle. Debe especificar los valores de estos atributos.

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks'
```

También puede especificar condiciones adicionales para la operación. La siguiente operación `DELETE` solo elimina el elemento si tiene más de 11 Awards (Premios).

```
DELETE FROM Music
WHERE Artist = 'Acme Band' AND SongTitle = 'PartiQL Rocks' AND Awards > 11
```

> **Note**
>
> Para obtener ejemplos de código mediante `DELETE` y `ExecuteStatement`, consulte Instrucciones de eliminación de PartiQL para DynamoDB.

Diferencias entre una base de datos relacional (SQL) y DynamoDB al eliminar una tabla

En SQL, se utiliza la instrucción `DROP TABLE` para eliminar una tabla. En Amazon DynamoDB, se utiliza la operación `DeleteTable`.

Temas

- Eliminación de una tabla con SQL
- Eliminación de una tabla en DynamoDB

Eliminación de una tabla con SQL

Cuando ya no necesita una tabla y se desea descartarla de forma permanente, se utilizaría la instrucción `DROP TABLE` de SQL.

```
DROP TABLE Music;
```

Después de eliminar una tabla, ya no se puede recuperar. En algunas bases de datos relacionales se permite deshacer una operación `DROP TABLE`, pero se trata de una funcionalidad específica del proveedor que no se implementa de manera generalizada.

Eliminación de una tabla en DynamoDB

En DynamoDB, `DeleteTable` es una operación similar. En el siguiente ejemplo, se elimina la tabla de forma permanente.

```
{
    TableName: "Music"
}
```

Recursos y herramientas de aprendizaje de Amazon DynamoDB

Puede utilizar los siguientes recursos adicionales para entender y trabajar con DynamoDB.

Temas

- Herramientas de programación y visualización
- Artículos de orientación prescriptiva
- Artículos del Centro de conocimientos
- Entradas de blog, repositorios y guías
- Presentaciones de modelado de datos y patrones de diseño
- Cursos de formación técnica

Herramientas de programación y visualización

Puede utilizar las siguientes herramientas de codificación y visualización para trabajar con DynamoDB:

- NoSQL Workbench para Amazon DynamoDB: una herramienta visual unificada que le ayuda a diseñar, crear, consultar y administrar tablas de DynamoDB. Proporciona características de modelado de datos, visualización de datos y desarrollo de consultas.
- Dynobase: una herramienta de escritorio que facilita la visualización de las tablas de DynamoDB y el trabajo con ellas, la creación de código de aplicación y la edición de registros con validación en tiempo real.
- Caja de herramientas de DynamoDB: un proyecto de Jeremy Daly que proporciona utilidades eficaces para trabajar con el modelado de datos y JavaScript y Node.js.
- Procesador de DynamoDB Streams: una herramienta sencilla que puede usar para trabajar con DynamoDB Streams.

Artículos de orientación prescriptiva

La orientación prescriptiva de AWS proporciona estrategias, guías y patrones comprobados a lo largo del tiempo para ayudar a acelerar los proyectos. Estos recursos los desarrollaron expertos en tecnología de AWS y la comunidad global de socios de AWS, en función de los años de experiencia ayudando a los clientes a alcanzar los objetivos comerciales.

Modelado y migración de datos

- Un modelo de datos jerárquico en DynamoDB
- Modelado de datos con DynamoDB
- Migrar una base de datos de Oracle a DynamoDB con AWS DMS

Tablas globales

- Uso de tablas globales de Amazon DynamoDB

sin servidor

- Implementar el patrón saga sin servidor con AWS Step Functions

Arquitectura SaaS

- Administrar los inquilinos de varios productos SaaS en un único plano de control

- Incorporación de inquilinos en la arquitectura SaaS para el modelo de silo mediante C# y AWS CDK

Protección de datos y movimiento de datos

- Configurar el acceso entre cuentas a Amazon DynamoDB
- Opciones completas de copia de tablas para DynamoDB
- Estrategia de recuperación de desastres para bases de datos en AWS

Misceláneo

- Ayudar a reforzar el etiquetado en DynamoDB

Tutoriales en vídeo de orientación prescriptiva

- Uso de la arquitectura sin servidor para crear canalizaciones de datos
- Novartis: Motor de compras: portal de adquisiciones impulsado por IA
- Veritiv: permita que la información haga la previsión de la demanda de ventas en lagos de datos de AWS
- mimik: aprovechamiento de la nube perimetral híbrida AWS para Support Edge Microservice Mesh
- Captura de datos de cambios con Amazon DynamoDB

Para ver más artículos y vídeos de orientación prescriptiva para DynamoDB, consulte Orientación prescriptiva.

Artículos del Centro de conocimientos

Los artículos y vídeos del Centro de conocimiento de AWS cubren las preguntas y solicitudes más frecuentes que recibimos de los clientes de AWS. A continuación se muestran algunos artículos actuales del Centro de conocimiento sobre tareas específicas relacionadas con DynamoDB:

Optimización de costos

- Cómo puedo optimizar los costes con Amazon DynamoDB?

Limitación y latencia

- How can I troubleshoot high latency on an Amazon DynamoDB table?
- Por qué se está limitando mi tabla de DynamoDB?
- Por qué se está limitando mi tabla de DynamoDB bajo demanda?

Paginación

- Cómo implemento la paginación en DynamoDB

Transacciones

- Por qué produce un error mi llamada a la API `TransactWriteItems` en DynamoDB

Solución de problemas

- Cómo resuelvo problemas con el escalado automático de DynamoDB?
- Cómo soluciono los errores de HTTP 4XX en DynamoDB

Para ver artículos y vídeos adicionales de DynamoDB, consulte los artículos del Centro de conocimiento.

Entradas de blog, repositorios y guías

Además de la Guía para desarrolladores de DynamoDB, hay muchos recursos útiles para trabajar con DynamoDB. Estas son algunas publicaciones de blog, repositorios y guías seleccionadas para trabajar con DynamoDB:

- Repositorio de AWS de Ejemplos de códigos DynamoDB en varios idiomas del SDK de AWS: Node.js, Java, Python, .Net, Go, y Rust.
- El libro de DynamoDB: una guía completa de Alex DeBrie que presenta un enfoque basado en la estrategia del modelado de datos con DynamoDB.
- Guía de DynamoDB: una guía abierta de Alex DeBrie que describe los conceptos básicos y las características avanzadas de la base de datos DynamoDB NoSQL.
- Cómo cambiar de RDBMS a DynamoDB en 20 sencillos pasos: una lista de pasos útiles para obtener información sobre el modelado de datos de Jeremy Daly.

- Hoja de trucos DynamoDB JavaScript DocumentClient: una hoja de trucos para ayudarle a empezar a crear aplicaciones con DynamoDB en un entorno Node.js o JavaScript.
- Vídeos de conceptos básicos de DynamoDB: esta lista de reproducción cubre muchos de los conceptos básicos de DynamoDB.

Presentaciones de modelado de datos y patrones de diseño

Puede utilizar los siguientes recursos sobre modelos de datos y patrones de diseño para aprovechar DynamoDB al máximo:

- AWS re:Invent 2019: Modelado de datos con DynamoDB
  - Una charla de Alex DeBrie que le presenta los principios del modelado de datos de DynamoDB.
- AWS re:Invent 2020: Modelado de datos con DynamoDB: Parte 1
- AWS re:Invent 2020: Modelado de datos con DynamoDB: Parte 2
- AWS re:Invent 2017: Advanced design patterns
- AWS re:Invent 2018: Advanced design patterns
- AWS re:Invent 2019: Advanced design patterns
  - Jeremy Daly comparte sus 12 ideas principales de esta sesión.
- AWS re:Invent 2020: DynamoDB advanced design patterns: Parte 1
- AWS re:Invent 2020: DynamoDB advanced design patterns: Parte 2
- Horario de oficina de DynamoDB en Twitch

> **Note**
>
> En cada sesión se explican diferentes casos de uso y ejemplos.

Cursos de formación técnica

Hay muchos cursos de formación y opciones educativas diferentes para obtener más información sobre DynamoDB. Estos son algunos ejemplos actuales:

- Desarrollo con Amazon DynamoDB: diseñado por AWS para guiarle de principiante a experto en el desarrollo de aplicaciones del mundo real con modelado de datos para Amazon DynamoDB.

- Curso de información detallada de DynamoDB: un curso de Pluralsight.
- Amazon DynamoDB: creación de aplicaciones controladas por bases de datos NoSQL: un curso del equipo de formación y certificación de AWS alojado en edX.

Lecturas y escrituras de DynamoDB

Las operaciones de lectura y escritura de DynamoDB se refieren a las operaciones que recuperan datos de una tabla (lecturas) e insertan, actualizan o eliminan datos en una tabla (escrituras). Cuando trabaja con DynamoDB, es fundamental comprender los conceptos de lectura y escritura, ya que afectan directamente al rendimiento y al costo de la aplicación.

En este tema se proporcionan detalles sobre los distintos tipos de coherencia de lectura que se aplican a DynamoDB. En este tema también se describe el consumo de unidades de las distintas operaciones de lectura y escritura que puede realizar.

Temas

- Coherencia de lectura de DynamoDB
- Operaciones de lectura y escritura de DynamoDB

Coherencia de lectura de DynamoDB

Amazon DynamoDB lee datos de tablas, índices secundarios locales (LSI), índices secundarios globales (GSI) y flujos. Para obtener más información, consulte Componentes básicos de Amazon DynamoDB. Tanto las tablas como los LSI ofrecen dos opciones de coherencia de lectura: coherente posterior (predeterminada) y altamente coherente. Todas las lecturas de los GSI y las secuencias son coherentes posteriores.

Cuando la aplicación escribe datos en una tabla de DynamoDB y recibe una respuesta HTTP 200 (OK), significa que la escritura se ha completado correctamente y ha persistido de forma duradera. DynamoDB proporciona aislamiento de lectura confirmada y garantiza que las operaciones de lectura siempre devuelvan valores confirmados para un elemento. La lectura nunca presentará una vista al elemento procedente de una escritura que finalmente no se ha realizado correctamente. El aislamiento de lectura confirmada no impide las modificaciones del elemento inmediatamente después de la operación de lectura.

Lecturas coherentes posteriores

“Coherente posterior” es el modelo de coherencia de lectura predeterminado para todas las operaciones de lectura. Al emitir lecturas coherentes posteriores a una tabla de DynamoDB o a un índice, es posible que las respuestas no reflejen los resultados de una operación de escritura

completada recientemente. Si repite la solicitud de lectura tras un breve intervalo de tiempo, la respuesta debería finalmente devolver el elemento más reciente. Se admiten lecturas coherentes posteriores en tablas, índices secundarios locales e índices secundarios globales. Tenga en cuenta que todas las lecturas de un flujo de DynamoDB también son coherentes posteriores.

Las lecturas coherentes posteriores cuestan la mitad que las lecturas altamente coherentes. Para obtener más información, consulte los precios de Amazon DynamoDB.

Lecturas altamente coherentes

Las operaciones de lectura como `GetItem`, `Query` y `Scan` proporcionan un parámetro `ConsistentRead` opcional. Si establece `ConsistentRead` a true, DynamoDB devuelve una respuesta con los datos más actualizados, de tal forma que refleja las actualizaciones de todas las operaciones de escritura anteriores que se han llevado a cabo correctamente. Las lecturas altamente coherentes solo se admiten en tablas e índices secundarios locales. No se admiten lecturas altamente coherentes desde un índice secundario global o un flujo de DynamoDB.

Coherencia de lectura de las tablas globales

DynamoDB también admite tablas globales para la replicación multiactiva y de varias regiones. Una tabla global se compone de varias tablas de réplicas en diferentes regiones de AWS. Cualquier cambio efectuado en cualquier elemento de cualquier tabla réplica se replica en todas las demás réplicas de la misma tabla global, normalmente en un segundo, y son coherentes posteriores. Para obtener más información, consulte Modos de coherencia.

Operaciones de lectura y escritura de DynamoDB

Las operaciones de lectura de DynamoDB permiten recuperar uno o más elementos de una tabla especificando el valor de la clave de partición y, de forma opcional, el valor de la clave de clasificación. Mediante las operaciones de escritura de DynamoDB, puede insertar, actualizar o eliminar elementos en una tabla. En este tema se explica el consumo de unidades de capacidad para estas dos operaciones.

Temas

- Consumo de unidades de capacidad para operaciones de lectura
- Consumo de unidades de capacidad para operaciones de escritura

Consumo de unidades de capacidad para operaciones de lectura

Las solicitudes de lectura de DynamoDB pueden ser altamente coherentes, coherentes posteriores o transaccionales.

- Una solicitud de lectura altamente coherente de un elemento de hasta 4 KB requiere una unidad de lectura.
- Una solicitud de lectura coherente posterior de un elemento de hasta 4 KB requiere media unidad de lectura.
- Una solicitud de lectura transaccional de un elemento de hasta 4 KB requiere dos unidades de lectura.

Para obtener más información sobre los modelos de consistencia de lectura de DynamoDB, consulte Coherencia de lectura de DynamoDB.

A efectos de las lecturas, los tamaños de los elementos se redondean al siguiente múltiplo de 4 KB. Por ejemplo, leer un elemento de 3 500 bytes consume el mismo rendimiento que leer un elemento de 4 KB.

Para leer un elemento mayor de 4 KB, DynamoDB necesita unidades de lectura adicionales. El número total de unidades de lectura necesarias depende del tamaño del elemento y de si se desea utilizar una lectura coherente posterior o una lectura altamente coherente. Por ejemplo, si el tamaño de su fila es de 8 KB, necesitará 2 unidades de lectura para realizar una lectura altamente coherente. Necesitará 1 unidad de lectura si elige lecturas coherentes posteriores o 4 unidades de lectura para una solicitud de lectura transaccional.

En la siguiente lista se describe cómo las operaciones de lectura de DynamoDB consumen unidades de lectura:

- GetItem: lee un solo elemento de una tabla. Para determinar el número de unidades que va a consumir `GetItem`, tome el tamaño del elemento y redondéelo al siguiente límite de 4 KB. Este es el número de unidades de lectura necesarias si se especifica una lectura altamente coherente. Si se trata de una lectura coherente posterior, que es el valor predeterminado, divida este número por dos.

  Por ejemplo, si lee un elemento de 3,5 KB, DynamoDB redondea su tamaño a 4 KB. Si lee un elemento de 10 KB, DynamoDB redondea su tamaño a 12 KB.

- BatchGetItem: lee hasta 100 elementos de una o varias tablas. DynamoDB procesa cada elemento del lote como una solicitud `GetItem` individual. DynamoDB redondea primero el tamaño de cada elemento al siguiente límite de 4 KB y, a continuación, calcula el tamaño total. El resultado no es necesariamente equivalente al tamaño total de todos los elementos. Por ejemplo, si `BatchGetItem` lee dos elementos de 1,5 KB y 6,5 KB, DynamoDB calcula el tamaño como 12 KB (4 KB + 8 KB). DynamoDB no calcula el tamaño como 8 KB (1,5 KB + 6,5 KB).
- Query: lee varios elementos que tienen el mismo valor de clave de partición. Todos los elementos devueltos se tratan como una sola operación de lectura, de tal forma que DynamoDB calcula el tamaño total de todos los elementos. A continuación, DynamoDB redondea el tamaño al siguiente límite de 4 KB. Por ejemplo, supongamos que la consulta devuelve 10 elementos cuyo tamaño combinado es de 40,8 KB. DynamoDB redondea el tamaño del elemento de la operación a 44 KB. Si una consulta devuelve 1500 elementos de 64 bytes cada uno, el tamaño acumulado es de 96 KB.
- Scan: lee todos los elementos de una tabla. DynamoDB considera el tamaño de los elementos que se evalúan, no el tamaño de los elementos que el examen devuelve. Para obtener más información sobre las operaciones Scan, consulte Análisis de tablas en DynamoDB.

> **Important**
>
> Si realiza una operación de lectura en un elemento que no existe, DynamoDB seguirá consumiendo rendimiento de lectura como se ha indicado anteriormente. En el caso de las operaciones `Query`/`Scan`, se le seguirá cobrando un rendimiento de lectura adicional basado en la coherencia de lectura y en el número de particiones que se han buscado para atender la solicitud, aunque no existan datos.

Para cualquier operación que devuelve elementos, puede solicitar un subconjunto de atributos para recuperarlos. Sin embargo, esto no afecta al cálculo del tamaño de los elementos. Por otra parte, `Query` y `Scan` pueden devolver recuentos de elementos en lugar de valores de atributos. Para obtener el recuento de los elementos, se consume la misma cantidad de unidades de lectura y se llevan a cabo los mismos cálculos de tamaño de los elementos. Esto se debe a que DynamoDB tiene que leer cada elemento para poder incrementar el recuento.

Consumo de unidades de capacidad para operaciones de escritura

Una unidad de escritura equivale a una escritura para un elemento con un tamaño de hasta 1 KB. Para escribir un elemento mayor de 1 KB, DynamoDB tendrá que consumir unidades de escritura adicionales. Las solicitudes de escritura transaccionales requieren 2 unidades de escritura para realizar una escritura para elementos de hasta 1 KB. El número total de unidades de solicitud de escritura necesarias depende del tamaño del elemento. Por ejemplo, si el tamaño de elemento es de 2 KB, se necesitan 2 unidades de lectura para mantener una solicitud de escritura o 4 unidades de escritura para una solicitud de escritura transaccional.

A efectos de las escrituras, los tamaños de los elementos se redondean al siguiente múltiplo de 1 KB. Por ejemplo, escribir un elemento de 500 bytes consume el mismo rendimiento que leer un elemento de 1 KB.

En la siguiente lista se describe cómo consumen unidades de escritura las operaciones de escritura de DynamoDB:

- PutItem: escribe un solo elemento en una tabla. Si ya existe un elemento con la misma clave principal en la tabla, la operación lo sustituye. Para calcular el consumo de desempeño provisionado, el tamaño de elemento que se tiene en cuenta es el mayor de los dos.
- UpdateItem: modifica un solo elemento en la tabla. DynamoDB considera el tamaño del elemento tal y como aparece antes y después de la actualización. El desempeño provisionado consumido refleja el mayor de estos tamaños de elemento. Aunque se actualice un subconjunto de atributos del elemento, `UpdateItem` consumirá la cantidad total de rendimiento aprovisionado (el mayor de los tamaños de elemento de “antes” y “después”).
- DeleteItem: elimina un solo elemento de una tabla. El consumo de desempeño provisionado se basa en el tamaño del elemento eliminado.
- BatchWriteItem: escribe hasta 25 elementos en una o varias tablas. DynamoDB procesa cada elemento del lote como una consulta `PutItem` o `DeleteItem` individual (las actualizaciones no son compatibles). DynamoDB redondea primero el tamaño de cada elemento al siguiente límite de 1 KB y, a continuación, calcula el tamaño total. El resultado no es necesariamente equivalente al tamaño total de todos los elementos. Por ejemplo, si `BatchWriteItem` escribe dos elementos de 500 bytes y 3,5 KB, DynamoDB calcula el tamaño como 5 KB (1 KB + 4 KB). DynamoDB no calcula un tamaño de 4 KB (500 bytes + 3,5 KB).

A efectos de las operaciones `PutItem`, `UpdateItem` y `DeleteItem`, DynamoDB redondea los tamaños de los elementos al 1 KB inmediatamente superior. Por ejemplo, si coloca o elimina un elemento de 1,6 KB, DynamoDB redondeará su tamaño a 2 KB.

Las operaciones `PutItem`, `UpdateItem` y `DeleteItem` permiten escrituras condicionales, en las que se especifica una expresión que debe evaluarse en true para que la operación se lleve a cabo correctamente. Aunque el resultado de evaluar la expresión sea false, DynamoDB consume unidades de capacidad de escritura de la tabla. La cantidad de unidades de capacidad de escritura consumidas depende del tamaño del elemento. Este elemento puede ser un elemento existente en la tabla o uno nuevo que esté intentando crear o actualizar. Por ejemplo, supongamos que un elemento existente tiene 300 KB. El nuevo elemento que está intentando crear o actualizar tiene 310 KB. Las unidades de capacidad de escritura consumidas serán de 310 KB para el nuevo elemento.

Capacidad de rendimiento de DynamoDB

En esta sección se ofrece una descripción general de los dos modos de rendimiento que hay disponibles para la tabla de DynamoDB y se explica cómo seleccionar el modo de capacidad adecuado para la aplicación. El modo de rendimiento de una tabla determina cómo se administra la capacidad de ella. El modo de rendimiento también determina cómo se le cobran las operaciones de lectura y escritura en las tablas. En Amazon DynamoDB, puede elegir entre el modo bajo demanda y el modo aprovisionado para sus tablas con el fin de adaptarse a los diferentes requisitos de carga de trabajo.

Temas

- Modo bajo demanda
- Modo aprovisionado
- Modo de capacidad bajo demanda de DynamoDB
- Modo de capacidad aprovisionada de DynamoDB
- Descripción del rendimiento en caliente de DynamoDB
- Capacidad de ampliación y de adaptación de DynamoDB
- Aspectos a tener en cuenta al cambiar los modos de capacidad en DynamoDB

Modo bajo demanda

El modo bajo demanda de Amazon DynamoDB es una opción de rendimiento sin servidor que simplifica la administración de bases de datos y se escala automáticamente para respaldar las aplicaciones más exigentes de los clientes. DynamoDB bajo demanda le permite crear una tabla sin preocuparse de la planificación de capacidad, supervisar el uso ni configurar políticas de escalado. DynamoDB bajo demanda ofrece precios de pago por solicitud para las solicitudes de lectura y escritura. De este modo, únicamente tendrá que pagar por aquello que utilice. Para tablas en modo en diferido, no necesita especificar el rendimiento de lectura y escritura que espera de su aplicación.

El modo bajo demanda es la opción de rendimiento predeterminada y recomendada para la mayoría de las cargas de trabajo de DynamoDB. DynamoDB se encarga de todos los aspectos de la administración del rendimiento y el escalado para admitir cargas de trabajo que pueden empezar siendo pequeñas y escalar hasta millones de solicitudes por segundo. Puede leer y escribir en las tablas de DynamoDB según sea necesario sin administrar la capacidad de rendimiento de la tabla. Para obtener más información, consulte Modo de capacidad bajo demanda de DynamoDB.

Modo aprovisionado

En el modo aprovisionado, debe especificar el número de lecturas y escrituras por segundo que necesita para la aplicación. Se le cobrará en función de la capacidad de lectura y escritura por hora que haya aprovisionado, no de la cantidad de esa capacidad aprovisionada que haya consumido realmente. Esto le ayuda a controlar el uso de DynamoDB para que se mantenga igual o menor que la velocidad de solicitudes definida y, de esta forma, poder predecir los costos.

Puede elegir utilizar capacidad aprovisionada si tiene cargas de trabajo estables con un crecimiento predecible y si puede prever con fiabilidad las necesidades de capacidad de la aplicación. Para obtener más información, consulte Modo de capacidad aprovisionada de DynamoDB.

Modo de capacidad bajo demanda de DynamoDB

Amazon DynamoDB bajo demanda ofrece una verdadera experiencia de base de datos sin servidor que se escala de forma automática para adaptarse a las cargas de trabajo más exigentes sin necesidad de planificación de capacidad. Bajo demanda simplifica el proceso de configuración, elimina la administración y supervisión de la capacidad y proporciona un escalado rápido y automático. Con los precios de pago por solicitud, no tiene que preocuparse por la capacidad inactiva porque solo paga por el rendimiento que realmente utiliza. Se le factura por solicitud de lectura o escritura, por lo que los costos reflejan directamente el uso real.

Al elegir el modo en diferido, DynamoDB se adapta de forma instantánea a sus cargas de trabajo a medida que aumentan o disminuyen a cualquier nivel de tráfico alcanzado previamente. Si el nivel de tráfico de una carga de trabajo alcanza un nuevo pico, DynamoDB se escala automáticamente para adaptarse al aumento de los requisitos de rendimiento. El modo bajo demanda es la opción de rendimiento predeterminada y recomendada porque simplifica la creación de aplicaciones modernas y sin servidor que pueden empezar siendo pequeñas y escalar hasta millones de solicitudes por segundo. Una vez escalada horizontalmente la tabla bajo demanda, podrá volver a alcanzar al instante el mismo rendimiento en el futuro sin limitaciones. Si no dirige tráfico alguno a la tabla, con el sistema bajo demanda no se le cobrará ningún rendimiento. Para obtener más información sobre las propiedades de escalado del modo bajo demanda, consulte Rendimiento inicial y propiedades de escalado.

Las tablas que utilizan el modo bajo demanda ofrecen la misma latencia de un milisegundo, acuerdo de nivel de servicio (SLA) y seguridad que el modo aprovisionado de DynamoDB.

> **Note**
>
> De forma predeterminada, DynamoDB lo protege del uso incontrolado e imprevisto. Para escalar más allá de los límites de rendimiento de lectura y escritura en el nivel de tabla de 40 000 para todas las tablas de la cuenta, puede solicitar un aumento de esta cuota. Las solicitudes de rendimiento que superan la cuota de rendimiento predeterminada de la tabla están limitadas. Para obtener más información, consulte Cuotas de rendimiento predeterminadas.

Si lo desea, también puede configurar el rendimiento máximo de lectura o escritura (o ambos) por segundo para tablas bajo demanda individuales e índices secundarios globales. Al configurar el rendimiento, puede limitar el uso y los costos de las tablas, protegerse contra un aumento no intencionado de los recursos consumidos y evitar el uso excesivo para que la administración de los costos sea predecible. Las solicitudes de rendimiento que superan el rendimiento máximo de la tabla tienen aplicada una limitación. Puede modificar el rendimiento máximo específico de la tabla en cualquier momento en función de los requisitos de su aplicación. Para obtener más información, consulte Rendimiento máximo de DynamoDB para las tablas bajo demanda.

Para empezar, cree o actualice una tabla para utilizar el modo bajo demanda. Para obtener más información, consulte Operaciones básicas en tablas de DynamoDB.

Puede cambiar las tablas del modo de capacidad aprovisionada al modo bajo demanda hasta cuatro veces en un periodo continuo de 24 horas. Las tablas pueden cambiar del modo bajo demanda al modo de capacidad aprovisionada en cualquier momento.

Para obtener más información sobre el cambio entre los modos de capacidad de lectura y escritura, consulte Aspectos a tener en cuenta al cambiar los modos de capacidad en DynamoDB. Para ver las cuotas de las tablas bajo demanda, consulte Rendimiento de lectura o escritura.

Temas

- Unidades de solicitud de lectura y de escritura
- Rendimiento inicial y propiedades de escalado
- Rendimiento máximo de DynamoDB para las tablas bajo demanda

Unidades de solicitud de lectura y de escritura

DynamoDB le cobra por las lecturas y escrituras que realiza su aplicación en sus tablas por unidades de solicitud de lectura y unidades de solicitud de escritura.

Una unidad de solicitud de lectura representa una lectura altamente coherente por segundo, o dos lecturas coherentes posteriores por segundo, para elementos con un tamaño máximo de 4 KB. Para obtener más información sobre los modelos de consistencia de lectura de DynamoDB, consulte Coherencia de lectura de DynamoDB.

Una unidad de solicitud de escritura representa una operación de escritura por segundo para un elemento con un tamaño máximo de 1 KB.

Para obtener más información sobre cómo se consumen las unidades de lectura y escritura, consulteOperaciones de lectura y escritura de DynamoDB.

Rendimiento inicial y propiedades de escalado

Las tablas de DynamoDB que utilizan el modo de capacidad bajo demanda se adaptan automáticamente al volumen de tráfico de la aplicación. Las nuevas tablas bajo demanda podrán soportar hasta 4000 escrituras por segundo y 12 000 lecturas por segundo. El modo de capacidad bajo demanda acomoda al instante hasta el doble del tráfico máximo alcanzado previamente en una tabla. Por ejemplo, supongamos que el patrón de tráfico de su aplicación oscila entre 25 000 y 50 000 lecturas altamente coherentes por segundo. El pico de tráfico anterior es de 50 000 lecturas por segundo. El modo de capacidad bajo demanda se adapta instantáneamente a un tráfico sostenido de hasta 100 000 lecturas por segundo. Si su aplicación soporta un tráfico de 100 000 lecturas por segundo, ese pico se convierte en su nuevo pico anterior. Este pico anterior permite que el tráfico posterior alcance hasta 200 000 lecturas por segundo.

Si su carga de trabajo genera más del doble que su pico anterior en una tabla, DynamoDB asigna automáticamente más capacidad a medida que aumenta su volumen de tráfico. Esta asignación de capacidad ayuda a garantizar que no se aplique una limitación en su carga de trabajo. Sin embargo, esta limitación controlada podría producirse si supera el doble del pico anterior en el plazo de 30 minutos. Por ejemplo, supongamos que el patrón de tráfico de su aplicación oscila entre 25 000 y 50 000 lecturas altamente coherentes por segundo. El pico de tráfico alcanzado anteriormente es de 50 000 lecturas por segundo. Le recomendamos que precaliente la tabla o espacie el crecimiento de su tráfico durante al menos 30 minutos antes de producir más de 100 000 lecturas por segundo. Para obtener más información acerca del precalentamiento, consulte Descripción del rendimiento en caliente de DynamoDB.

DynamoDB no establece la restricción de limitación de 30 minutos si el pico de tráfico de la carga de trabajo se mantiene dentro del doble del pico anterior. Si el pico de tráfico supera el doble de ese pico, asegúrese de que este aumento se produce 30 minutos después de la última vez que alcanzó el pico.

Rendimiento máximo de DynamoDB para las tablas bajo demanda

En el caso de las tablas bajo demanda, si lo desea, puede especificar el rendimiento máximo de lectura o escritura (o ambos) por segundo en tablas individuales y en los índices secundarios globales (GSI) asociados. Especificar un rendimiento máximo bajo demanda ayuda a limitar el uso y los costos de las tablas. De forma predeterminada, no se aplica la configuración de rendimiento máximo y la tasa de rendimiento bajo demanda está limitada por la cuota de servicio de AWS de rendimiento de lectura y escritura en el nivel de tabla de 40 000 para todas las tablas de la cuenta. Si es necesario, también puede solicitar un aumento de la cuota de servicio.

Al configurar el rendimiento máximo para una tabla bajo demanda, se aplicará una limitación a las solicitudes de rendimiento que superen la cantidad máxima especificada. Puede modificar la configuración del rendimiento de las tablas en cualquier momento en función de los requisitos de su aplicación.

Estos son algunos casos de uso comunes que pueden beneficiarse del uso de un rendimiento máximo para las tablas bajo demanda:

- Optimización de los costos de rendimiento: el uso del rendimiento máximo para las tablas bajo demanda permite disponer de un nivel adicional de previsibilidad de los costos y facilidad de administración. Además, ofrece una mayor flexibilidad para usar el modo bajo demanda para soportar cargas de trabajo con diferentes patrones de tráfico y presupuestos.
- Protección contra el uso excesivo: al establecer el rendimiento máximo, puede evitar un aumento accidental del consumo de lecturas o escrituras, que podría deberse a un código no optimizado o a procesos fraudulentos en una tabla bajo demanda. Esta configuración de tabla puede evitar que las organizaciones consuman recursos excesivos en un período de tiempo determinado.
- Protección de los servicios posteriores: la aplicación de un cliente puede incluir tecnologías sin servidor y con servidor. La parte sin servidor de la arquitectura sin puede escalarse rápidamente para adaptarse a la demanda. Sin embargo, los componentes posteriores con capacidades fijas podrían saturarse. La implementación de una configuración de rendimiento máximo para las tablas bajo demanda puede evitar que un gran volumen de eventos se propague a varios componentes posteriores y se produzcan efectos secundarios inesperados.

Puede configurar el rendimiento máximo del modo bajo demanda en tablas nuevas y existentes de una sola región, así como en tablas globales y GSI. También puede configurar el rendimiento máximo durante la restauración de tablas y la importación de datos desde los flujos de trabajo de Amazon S3.

Puede especificar la configuración de rendimiento máximo de una tabla bajo demanda mediante la consola de DynamoDB, la AWS CLI, AWS CloudFormation o la API de DynamoDB.

> **Note**
>
> El rendimiento máximo de una tabla bajo demanda se aplica en la medida de lo posible y debe considerarse como un objetivo en lugar de un límite de solicitudes garantizado. Es posible que su carga de trabajo supere temporalmente el rendimiento máximo especificado debido a la capacidad de ampliación. En algunos casos, DynamoDB utiliza la capacidad de ampliación para atender las lecturas o escrituras que superan la configuración de rendimiento máxima de la tabla. Con la capacidad de ráfaga, pueden realizarse correctamente solicitudes de lectura o escritura inesperadas que, de otro modo, habrían sido objeto de una limitación controlada.

Temas

- Consideraciones al utilizar el rendimiento máximo en el modo bajo demanda
- Limitación de solicitudes y métricas de CloudWatch

Consideraciones al utilizar el rendimiento máximo en el modo bajo demanda

Cuando se utiliza el rendimiento máximo para las tablas en el modo bajo demanda, hay que tener en cuenta lo siguiente:

- Puede establecer de forma independiente el rendimiento máximo de lectura y escritura para cualquier tabla bajo demanda o para un índice secundario global individual dentro de esa tabla para ajustar su enfoque a los requisitos específicos.
- Puede utilizar Amazon CloudWatch para monitorear y conocer las métricas de uso de tabla de DynamoDB y para determinar la configuración de rendimiento máximo adecuada para el modo bajo demanda. Para obtener más información, consulte Dimensiones y métricas de DynamoDB.
- Al especificar la configuración de rendimiento máximo de lectura o escritura (o ambas) en una sola réplica de tabla global, esa misma configuración se aplica automáticamente a todas las

tablas de réplicas. Es importante que las tablas de réplicas y los índices secundarios de la tabla global tengan una configuración de rendimiento de escritura idéntica para garantizar la replicación adecuada de los datos. Para obtener más información, consulte Prácticas recomendadas para tablas globales.

- El rendimiento máximo de lectura o escritura más bajo que puede especificar es una unidad de solicitud por segundo.
- El rendimiento máximo que especifique debe ser inferior a la cuota de rendimiento predeterminada que está disponible para cualquier tabla bajo demanda o índice secundario global individual dentro de esa tabla.

Limitación de solicitudes y métricas de CloudWatch

Si la aplicación supera el rendimiento máximo de lectura o escritura que ha establecido en la tabla bajo demanda, DynamoDB comienza a aplicar límites a esas solicitudes. Cuando DynamoDB aplica una limitación controlada a una lectura o escritura, devuelve una `ThrottlingException` a la persona que llama. A continuación, puede tomar las medidas adecuadas, si es necesario. Por ejemplo, puede aumentar o deshabilitar la configuración de rendimiento máximo de la tabla o esperar un poco antes de volver a intentar la solicitud.

Para simplificar el monitoreo del rendimiento máximo configurado para una tabla o un índice secundario global, CloudWatch proporciona las siguientes métricas: OnDemandMaxReadRequestUnits y OnDemandMaxWriteRequestUnits.

Modo de capacidad aprovisionada de DynamoDB

Al crear una nueva tabla aprovisionada en DynamoDB, debe especificar su capacidad de rendimiento aprovisionada. Esta es la cantidad de rendimiento de lectura y escritura que puede soportar la tabla. Se le cobrará en función de la capacidad de lectura y escritura por hora que haya aprovisionado, no de la cantidad de esa capacidad aprovisionada que haya consumido realmente.

A medida que cambian los requisitos de datos y acceso de la aplicación, puede que tenga que ajustar la configuración de rendimiento de la tabla. Puede utilizar el escalado automático para ajustar automáticamente la capacidad aprovisionada de la tabla en respuesta a los patrones de tráfico. El escalado automático de DynamoDB utiliza una política de escalado en Escalado automático de aplicaciones. Para configurar el escalado automático en DynamoDB, debe establecer los niveles mínimo y máximo de capacidad de lectura y escritura, además del porcentaje de utilización objetivo.

Auto Scaling de aplicaciones crea y administra las alarmas de CloudWatch que desencadenan eventos de escalado cuando la métrica se desvía del destino. El escalado automático supervisa la actividad de la tabla y aumenta o reduce la configuración de capacidad en función de los umbrales preconfigurados. El escalado automático se activa cuando la capacidad consumida supera el objetivo de utilización configurado durante dos minutos consecutivos. Es posible que las alarmas de CloudWatch tarden unos minutos antes de desencadenar el escalado automático. Para obtener más información, consulte Administración automática de la capacidad de rendimiento con la función Auto Scaling de DynamoDB.

Si utiliza la función Auto Scaling de DynamoDB, la configuración de rendimiento se modificarán automáticamente en respuesta a las cargas de trabajo reales. También puede usar la operación UpdateTable para ajustar manualmente la capacidad de rendimiento de la tabla. Por ejemplo, es posible que prefiera hacerlo de ese modo para cargar datos masivamente de un almacén de datos en la nueva tabla de DynamoDB. Podría crear la tabla con un ajuste de desempeño de escritura mayor y, a continuación, reducir este ajuste una vez finalizada la carga masiva de datos.

> **Note**
>
> De forma predeterminada, DynamoDB lo protege del uso incontrolado e imprevisto. Para escalar más allá de los límites de rendimiento de lectura y escritura en el nivel de tabla de 40 000 para todas las tablas de la cuenta, puede solicitar un aumento de esta cuota. Las solicitudes de rendimiento que superan la cuota de rendimiento predeterminada de la tabla están limitadas. Para obtener más información, consulte Cuotas de rendimiento predeterminadas.

Puede cambiar las tablas del modo de capacidad aprovisionada al modo bajo demanda hasta cuatro veces en un periodo continuo de 24 horas. Las tablas pueden cambiar del modo bajo demanda al modo de capacidad aprovisionada en cualquier momento.

Para obtener más información sobre el cambio entre los modos de capacidad de lectura y escritura, consulte Aspectos a tener en cuenta al cambiar los modos de capacidad en DynamoDB.

Temas

- Unidades de capacidad de lectura y de escritura
- Elección de los ajustes de rendimiento iniciales
- Escalado automático de DynamoDB

- Administración automática de la capacidad de rendimiento con la función Auto Scaling de DynamoDB
- Capacidad reservada de DynamoDB

Unidades de capacidad de lectura y de escritura

Para las tablas en el modo aprovisionado, especifique los requisitos de rendimiento en unidades de capacidad. Estas unidades representan la cantidad de datos que la aplicación necesita para leer o escribir por segundo. Puede cambiar esta configuración más adelante, si es necesario, o habilitar la función Auto Scaling de DynamoDB para que se modifiquen automáticamente.

Para un elemento de hasta 4 KB, una unidad de capacidad de lectura (RCU) representa una operación de lectura altamente coherente por segundo o dos operaciones de lectura coherente posterior por segundo. Para obtener más información sobre los modelos de consistencia de lectura de DynamoDB, consulte Coherencia de lectura de DynamoDB.

Una unidad de capacidad de escritura (WCU) equivale a una escritura por segundo para un elemento con un tamaño máximo de 1 KB. Para obtener más información sobre las diferentes operaciones de lectura y escritura, consulte Operaciones de lectura y escritura de DynamoDB.

Elección de los ajustes de rendimiento iniciales

Cada aplicación tiene requisitos diferentes de lectura y escritura en una base de datos. Al determinar la configuración de rendimiento inicial de una tabla de DynamoDB, debe tener en cuenta lo siguiente:

- Velocidades esperadas de solicitudes de lectura y escritura: debe calcular la cantidad de lecturas y escrituras que debe realizar por segundo.
- Tamaños de los elementos: algunos elementos son lo bastante pequeños para leerlos o escribirlos con una sola unidad de capacidad. Los elementos mayores requieren varias unidades de capacidad. Si calcula el tamaño medio de los elementos que contendrá la tabla, podrá especificar una configuración precisa del rendimiento aprovisionado de la tabla.
- Requisitos de consistencia de lectura: las unidades de capacidad de lectura se basan en operaciones de lectura altamente coherente, que consumen el doble de recursos de la base de datos que las lecturas coherentes posteriores. Es importante determinar si la aplicación necesita las lecturas de consistencia alta o si es posible adoptar un enfoque más flexible que realice en su lugar lecturas consistentes finales. Las operaciones de lectura en DynamoDB son coherentes posteriores de manera predeterminada. Si es preciso, puede solicitar lecturas altamente coherentes para estas operaciones.

Por ejemplo, supongamos que desea leer 80 elementos por segundo de una tabla. Los elementos tienen un tamaño de 3 KB y desea realizar lecturas altamente coherentes. En este caso, cada lectura requiere una unidad de capacidad de lectura aprovisionada. Para determinar esta cifra, hay que dividir el tamaño de elemento de la operación por 4 KB. A continuación, se redondea al número entero más próximo, como se muestra en el siguiente ejemplo:

- 3 KB / 4 KB = 0,75 o 1 unidad de capacidad de lectura

Por lo tanto, para leer 80 elementos por segundo de una tabla, establezca el rendimiento de lectura aprovisionado de la tabla en 80 unidades de capacidad de lectura, como se muestra en el siguiente ejemplo:

- 1 unidad de capacidad de lectura por elemento × 80 lecturas por segundo = 80 unidades de capacidad de lectura

Ahora, suponga que desea escribir 100 elementos por segundo en la tabla y que los elementos tienen un tamaño de 512 bytes. En este caso, cada escritura requiere una unidad de capacidad de escritura aprovisionada. Para determinar esta cifra, hay que dividir el tamaño de elemento de la operación por 1 KB. A continuación, se redondea al número entero más próximo, como se muestra en el siguiente ejemplo:

- 512 bytes /1 KB = 0,5 o 1 unidad de capacidad de escritura

Para escribir 100 elementos por segundo en la tabla, establezca el rendimiento de escritura aprovisionado para la tabla en 100 unidades de capacidad de escritura:

- 1 unidad de capacidad de escritura por elemento × 100 escrituras por segundo = 100 unidades de capacidad de escritura

Escalado automático de DynamoDB

El escalado automático de DynamoDB administra activamente la capacidad de rendimiento aprovisionada de las tablas y los índices secundarios globales. Con Auto Scaling, se define un rango (los límites superior e inferior) de unidades de capacidad de lectura y escritura. También se definir un porcentaje de objetivo de utilización comprendido en ese rango. La función Auto Scaling de DynamoDB intenta mantener el objetivo de utilización aunque la carga de trabajo de la aplicación aumente o disminuya.

Con el escalado automático de DynamoDB, una tabla o un índice secundario global pueden aumentar su capacidad de lectura y escritura aprovisionada para hacer frente a los aumentos repentinos de tráfico, sin que se aplique la limitación controlada a las solicitudes. Cuando la carga de trabajo disminuye, el escalado automático de DynamoDB puede reducir el rendimiento para evitar que tenga que pagar por una capacidad aprovisionada que no se utiliza.

> **Note**
>
> Si usa la Consola de administración de AWS para crear una tabla o un índice secundario global, la función Auto Scaling de DynamoDB se habilita de forma predeterminada. Puede administrar la configuración de escalado automático en cualquier momento mediante la consola, la AWS CLI o uno de los AWS. Para obtener más información, consulte Administración automática de la capacidad de rendimiento con la función Auto Scaling de DynamoDB.

Tasa de utilización

La tasa de utilización puede ayudarle a determinar si su capacidad de aprovisionamiento es excesiva, en cuyo caso debería reducir la capacidad de las tablas para ahorrar costos. Por el contrario, también puede ayudarle a determinar si su capacidad de aprovisionamiento es insuficiente. En este caso, debería aumentar la capacidad de las tablas para evitar que se apliquen limitaciones en las solicitudes en casos inesperados de tráfico intenso. Para obtener más información, consulte Amazon DynamoDB auto scaling: Performance and cost optimization at any scale.

Si utiliza el escalado automático de DynamoDB, también tendrá que establecer un porcentaje de utilización objetivo. El escalado automático utilizará este porcentaje como objetivo para ajustar la capacidad arriba o abajo. Recomendamos establecer el objetivo de utilización en un 70 %. Para obtener más información, consulte Administración automática de la capacidad de rendimiento con la función Auto Scaling de DynamoDB.

Administración automática de la capacidad de rendimiento con la función Auto Scaling de DynamoDB

Muchas cargas de trabajo de base de datos son de naturaleza cíclica, mientras que otras son difíciles de predecir con antelación. Por ejemplo, tomemos una aplicación de redes sociales en la que la mayoría de los usuarios están activos en el horario diurno. La base de datos debe satisfacer

los requisitos de la actividad diurna, pero no se requieren los mismos niveles de rendimiento por la noche. Otro ejemplo: tomemos una nueva aplicación de juegos para móviles cuya adopción está siendo inesperadamente rápida. Si el juego adquiere demasiada popularidad, podría superar los recursos disponibles en la base de datos, lo que daría lugar a un rendimiento lento y a clientes descontentos. Estos tipos de cargas de trabajo suelen requerir intervención manual para escalar los recursos de la base de datos en sentido ascendente o descendente en respuesta a las variaciones en los niveles de uso.

El escalado automático de Amazon DynamoDB usa el servicio Auto Scaling de aplicaciones de AWS para ajustar de manera dinámica y automática la capacidad de rendimiento aprovisionada en respuesta a los patrones de tráfico reales. Esto permite a una tabla o índice secundario global (GSI) incrementar su capacidad de lectura y escritura aprovisionada para abastecer incrementos repentinos del tráfico sin limitaciones. Cuando la carga de trabajo disminuye, el Auto Scaling de aplicaciones puede reducir el rendimiento para evitar que tenga que pagar por una capacidad aprovisionada que no se utiliza.

> **Note**
>
> Si usa la Consola de administración de AWS para crear una tabla o un índice secundario global, la función Auto Scaling de DynamoDB se habilita de forma predeterminada. Puede modificar los ajustes de Auto Scaling en cualquier momento. Para obtener más información, consulte Uso de la Consola de administración de AWS con la función Auto Scaling de DynamoDB.
>
> Cuando elimina una tabla o una réplica de tabla global, los destinos escalables, las políticas de escalado o las alarmas de CloudWatch asociadas no se eliminan automáticamente con ella.

Con Auto Scaling de aplicaciones, puede crear una política de escalado para una tabla o un índice secundario global. La política de escalado especifica si desea escalar la capacidad de lectura o de escritura (o ambas), así como los ajustes de unidades de capacidad provisionada mínimas y máximas para la tabla o el índice.

La política de escalado contiene además un valor de objetivo de utilización, es decir, el porcentaje de rendimiento aprovisionado consumido en un momento dado. Auto Scaling de aplicaciones utiliza un algoritmo de seguimiento de destino para ajustar el rendimiento aprovisionado de la tabla (o el índice) al alza o a la baja en respuesta a las cargas de trabajo reales, de tal forma que la utilización de la capacidad real se mantenga en valores iguales o parecidos al objetivo de utilización.

Las salidas de DynamoDB consumieron el rendimiento aprovisionado durante periodos de un minuto. El escalado automático se activa cuando la capacidad consumida supera el objetivo de utilización configurado durante dos minutos consecutivos. Es posible que las alarmas de CloudWatch tarden unos minutos antes de desencadenar el escalado automático. Este retraso garantiza una evaluación precisa de las métricas de CloudWatch. Si los picos de rendimiento consumidos están separados por más de un minuto, es posible que no se desencadene el escalado automático. Del mismo modo, se puede producir un evento de reducción vertical cuando 15 puntos de datos consecutivos sean inferiores a la utilización objetivo. En cualquier caso, tras desencadenar el escalado automático, se invoca la API UpdateTable. La actualización de la capacidad aprovisionada para la tabla o el índice puede tardar unos minutos. Durante este periodo, cualquier solicitud que supere la capacidad aprovisionada previamente de las tablas se limita.

> **Important**
>
> No se puede ajustar el número de puntos de datos que se van a violar para desencadenar la alarma subyacente (aunque el número actual podría cambiar en el futuro).

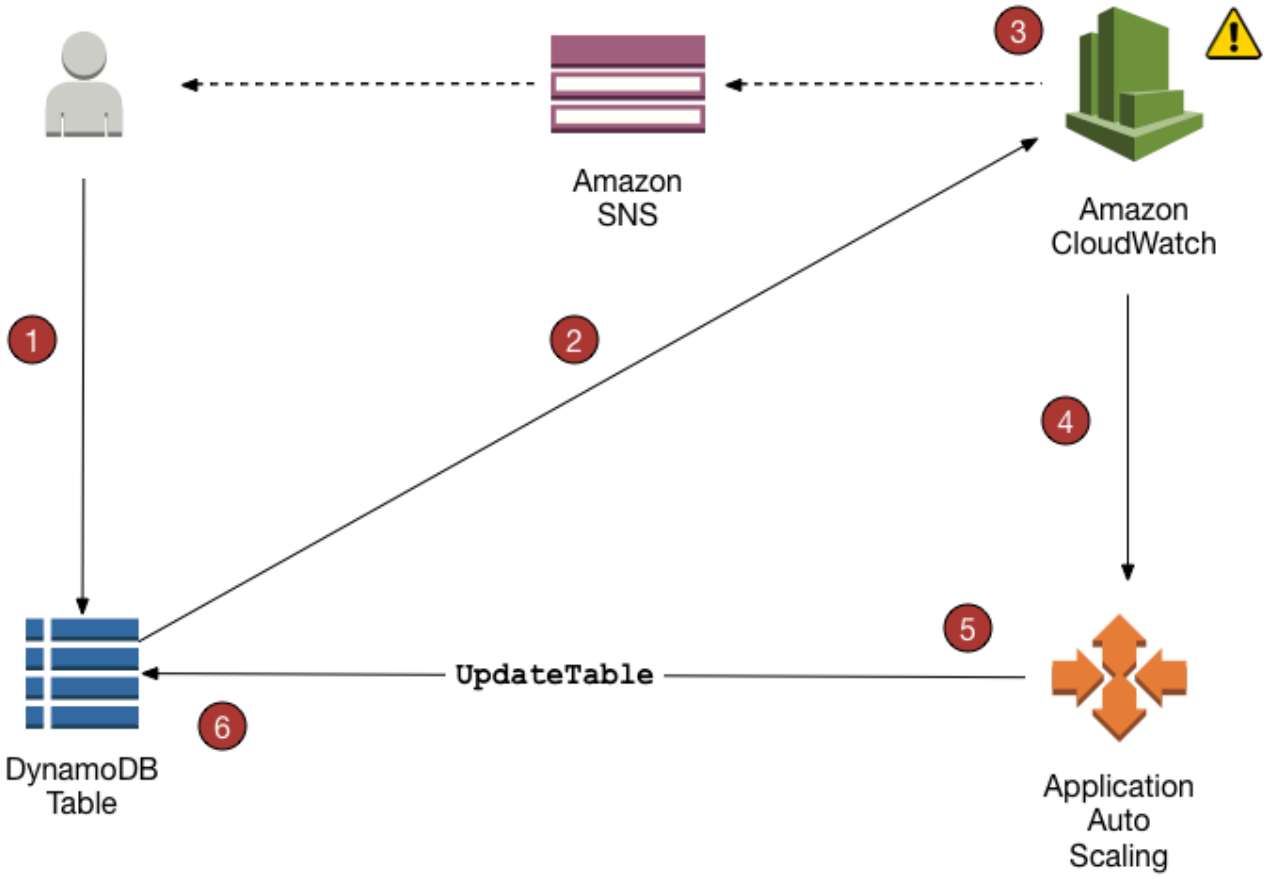
Puede establecer los valores de objetivo de utilización de escalado automático entre un 20 y un 90 por ciento de la capacidad de lectura y escritura.

> **Note**
>
> Además de con las tablas, el escalado automático de DynamoDB es compatible con los índices secundarios globales. Cada índice secundario global tiene su propia capacidad de rendimiento aprovisionada que es independiente de la de su tabla base. Al crear una política de escalado para un índice secundario global, Auto Scaling de aplicaciones ajusta los ajustes de rendimiento aprovisionado del índice para asegurarse de que el uso real se mantenga en valores iguales o parecidos al porcentaje de utilización deseado.

Funcionamiento de la función Auto Scaling de DynamoDB

> **Note**
>
> Para comenzar rápidamente a usar la función Auto Scaling de DynamoDB, consulte Uso de la Consola de administración de AWS con la función Auto Scaling de DynamoDB.

En el siguiente diagrama se ofrece información general sobre cómo el escalado automático de DynamoDB administra la capacidad de rendimiento de una tabla.



En los pasos siguientes se resume el proceso de Auto Scaling mostrado en el diagrama anterior:

1. Cree una política de Auto Scaling de aplicaciones para la tabla de DynamoDB.
2. DynamoDB publica métricas de capacidad consumida en Amazon CloudWatch.
3. Si la capacidad consumida de la tabla supera el objetivo de utilización (o no lo alcanza) durante un periodo de tiempo específico, Amazon CloudWatch activa una alarma. Puede ver la alarma en la consola y recibir notificaciones mediante Amazon Simple Notification Service (Amazon SNS).
4. La alarma de CloudWatch invoca a Auto Scaling de aplicaciones para evaluar la política de escalado.
5. Auto Scaling de aplicaciones emite una solicitud `UpdateTable` para ajustar el rendimiento aprovisionado de la tabla.
6. DynamoDB procesa la solicitud `UpdateTable` y aumenta (o reduce) dinámicamente la capacidad de rendimiento aprovisionada de la tabla para que sea parecida al objetivo de utilización.

Para comprender cómo funciona el escalado automático de DynamoDB, supongamos que tenemos una tabla denominada `ProductCatalog`. Es infrecuente que se realicen cargas masivas de datos en la tabla, de modo que presenta poca actividad de escritura. Sin embargo, sí experimenta una intensa actividad de lectura, que varía en cada momento. Gracias a las métricas de Amazon CloudWatch de `ProductCatalog` que se monitorean, ha determinado que la tabla requiere 1 200 unidades de capacidad de lectura (para evitar que DynamoDB aplique una limitación controlada a las solicitudes de lectura durante los picos de actividad). También ha determinado que `ProductCatalog` requiere como mínimo 150 unidades de capacidad de lectura, cuando el tráfico de lectura se encuentra en el punto más bajo. Para obtener más información sobre cómo prevenir la limitación, consulte Solución de problemas de limitación en Amazon DynamoDB.

Dentro del rango de 150 a 1200 unidades de capacidad de lectura, decide que un objetivo de utilización del 70 por ciento sería apropiado para la tabla `ProductCatalog`. El objetivo de utilización es la proporción de unidades de capacidad consumidas respecto de las unidades de capacidad provisionadas y se expresa como un porcentaje. Auto Scaling de aplicaciones utiliza el algoritmo de seguimiento de destino para asegurarse de que la capacidad de lectura provisionada de `ProductCatalog` se ajuste de acuerdo con las necesidades, de tal forma que la utilización permanezca próxima al 70 %.

> **Note**
>
> La función de escalado automático de DynamoDB modifica la configuración de rendimiento aprovisionado solo cuando la carga de trabajo real se mantiene elevada o reducida durante un periodo sostenido de varios minutos. El algoritmo de seguimiento de destino de Auto Scaling de aplicaciones intenta mantener el objetivo de utilización en el valor elegido o en valores próximos a él a largo plazo.
>
> Los picos de actividad repentinos y breves se atienden gracias a la capacidad de ampliación incorporada de la tabla. Para obtener más información, consulte Capacidad de ampliación.

Para habilitar el escalado automático de DynamoDB para la tabla `ProductCatalog`, debe crear una política de escalado. La política especifica los elementos siguientes:

- La tabla o el índice secundario global que desea administrar
- Qué tipo de capacidad va a administrar (capacidad de lectura o capacidad de escritura)
- Los límites superior e inferior de los ajustes de desempeño provisionado
- Su objetivo de utilización

Al crear una política de escalado, Auto Scaling de aplicaciones crea automáticamente un par de alarmas de Amazon CloudWatch. Cada par representa los límites superior e inferior de la configuración de rendimiento provisionado. Estas alarmas de CloudWatch se activan cuando la utilización real de la tabla se desvía del objetivo de utilización durante un periodo de tiempo prolongado.

Cuando se activa una de las alarmas de CloudWatch, Amazon SNS envía una notificación (si se ha habilitado). A continuación, la alarma de CloudWatch invoca a Auto Scaling de aplicaciones que, a su vez, notifica a DynamoDB para que ajuste la capacidad aprovisionada de la tabla `ProductCatalog` al alza o a la baja, según corresponda.

Durante un evento de escalado, la AWS Config se cobra por cada elemento de configuración registrado. Cuando se produce un evento de escalado, se crean cuatro alarmas de CloudWatch para cada evento de escalado automático de lectura y escritura: alarmas ProvisionedCapacity: ProvisionedCapacityLow, ProvisionedCapacityHigh y alarmas ConsumedCapacity: AlarmHigh, AlarmLow. Esto da como resultado un total de ocho alarmas. Por lo tanto, AWS Config registra ocho elementos de configuración para cada evento de escalado.

> **Note**
>
> También puede programar el escalado de DynamoDB para que se realice en determinados momentos. Descubra los pasos básicos aquí.

Notas de uso

Antes de comenzar a usar la función Auto Scaling de DynamoDB, debe tener en cuenta lo siguiente:

- La función Auto Scaling de DynamoDB puede aumentar la capacidad de lectura o escritura tan a menudo como sea preciso, de acuerdo con la política de Auto Scaling. Todas las cuotas de DynamoDB siguen vigentes, tal como se describe en Cuotas en Amazon DynamoDB.
- La función Auto Scaling de DynamoDB no le impide modificar manualmente la configuración de rendimiento aprovisionado. Estos ajustes manuales no afectan a las alarmas de CloudWatch vigentes relacionadas con la función Auto Scaling de DynamoDB.
- Si habilita la función Auto Scaling de DynamoDB en una tabla que tiene uno o varios índices secundarios globales, recomendamos encarecidamente aplicar también Auto Scaling de manera uniforme a esos índices. Esto contribuirá a garantizar un mejor rendimiento en las escrituras y lecturas de las tablas, y a evitar la limitación. Puede activar el escalado automático si selecciona

Apply same settings to global secondary indexes (Aplicar la misma configuración a los índices secundarios globales) en la Consola de administración de AWS. Para obtener más información, consulte Habilitación de la función Auto Scaling de DynamoDB en tablas existentes.

- Cuando elimina una tabla o una réplica de tabla global, los destinos escalables, las políticas de escalado o las alarmas de CloudWatch asociadas no se eliminan automáticamente con ella.
- Al crear un GSI para una tabla existente, la función Auto Scaling no está habilitada para el GSI. Tendrá que administrar manualmente la capacidad mientras se construye el GSI. Una vez que se complete el relleno del GSI y este alcance el estado activo, la función de escalado automático funcionará con normalidad.

Uso de la Consola de administración de AWS con la función Auto Scaling de DynamoDB

Si usa la Consola de administración de AWS para crear una tabla nueva, la función Auto Scaling de Amazon DynamoDB se habilita para esa tabla de forma predeterminada. También puede utilizar la consola para habilitar Auto Scaling en las tablas existentes, modificar la configuración de esta función o deshabilitarla.

> **Note**
>
> Para obtener características más avanzadas como la configuración de tiempos de recuperación de escalado y reducción horizontal, utilice la AWS Command Line Interface (AWS CLI) para administrar el Auto Scaling de DynamoDB. Para obtener más información, consulte Uso de la AWS CLI para administrar la función Auto Scaling de DynamoDB.

Temas

- Antes de comenzar: concesión de permisos a los usuarios para la función Auto Scaling de DynamoDB
- Creación de una nueva tabla con la función Auto Scaling habilitada
- Habilitación de la función Auto Scaling de DynamoDB en tablas existentes
- Visualización de las actividades de Auto Scaling en la consola
- Modificación o deshabilitación de la configuración de Auto Scaling de DynamoDB

Antes de comenzar: concesión de permisos a los usuarios para la función Auto Scaling de DynamoDB

En AWS Identity and Access Management (IAM), la política `DynamoDBFullAccess` administrada por AWS proporciona los permisos necesarios para utilizar la consola de DynamoDB. No obstante, para el escalamiento automático de DynamoDB, los usuarios necesitan permisos adicionales.

> **Important**
>
> Para eliminar una tabla habilitada para el escalado automático se necesitan permisos `application-autoscaling:*`. La política `DynamoDBFullAccess` administrada por AWS incluye estos permisos.

Para configurar un usuario para el acceso a la consola de DynamoDB y el escalamiento automático de DynamoDB, cree un rol y agregue la política AmazonDynamoDBFullAccess a dicho rol. A continuación, asigne el rol a un usuario.

Creación de una nueva tabla con la función Auto Scaling habilitada

> **Note**
>
> El escalamiento automático de DynamoDB requiere la presencia de un rol vinculado al servicio (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) que realice acciones de escalamiento automático en su nombre. Este rol se crea automáticamente. Para obtener más información, consulte Roles vinculados a servicios de Auto Scaling de aplicaciones en la Guía del usuario de Auto Scaling de aplicaciones.

Para crear una nueva tabla con la función Auto Scaling habilitada

1. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. Seleccione Create table.
3. En la página Crear tabla, ingrese los detalles de Nombre de tabla y clave principal.
4. Si elige Configuración predeterminada, el escalado automático se habilita en la nueva tabla.

   De lo contrario, elija Personalizar la configuración y haga lo siguiente para especificar la configuración personalizada de la tabla:

a. Para Clase de tabla, mantenga la selección predeterminada de DynamoDB Standard.

b. Para la Configuración de la capacidad de lectura o escritura, mantenga la selección predeterminada de Aprovisionada y, a continuación, haga lo siguiente:

   i. Para la Capacidad de lectura, asegúrese de que el Escalado automático esté configurado en Activado.

   ii. Para la Capacidad de escritura, asegúrese de que el Escalado automático esté configurado en Activado.

   iii. Para Capacidad de lectura y Capacidad de escritura, establezca la política de escalado que desee para la tabla y, opcionalmente, para todos los índices secundarios globales de la tabla.

   - Unidades de capacidad mínimas: introduzca el límite inferior del intervalo de escalamiento automático.
   - Unidades de capacidad máxima: introduzca el límite superior del intervalo de escalamiento automático.
   - Objetivo de utilización: introduzca su porcentaje de utilización objetivo para la tabla.

   > **Note**
   >
   > Si crea un índice secundario global para la nueva tabla, la capacidad del índice en el momento de la creación será la misma que la capacidad de la tabla base. Puede cambiar la capacidad del índice en la configuración de la tabla después de crearla.

5. Seleccione Create table (Creación de tabla). Esto crea la tabla con los parámetros de escalado automático especificados.

Habilitación de la función Auto Scaling de DynamoDB en tablas existentes

> **Note**
>
> El escalamiento automático de DynamoDB requiere la presencia de un rol vinculado al servicio (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) que realice

acciones de escalamiento automático en su nombre. Este rol se crea automáticamente. Para obtener más información, consulte Roles vinculados a servicios para Aplication Auto Scaling.

Para habilitar la función Auto Scaling de DynamoDB en una tabla existente

1. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación del lado izquierdo de la consola, elija Tablas.
3. Elija la tabla en la que desee habilitar el escalado automático y, a continuación, haga lo siguiente:

   a. Elija la pestaña Configuración adicional.

   b. En la sección Capacidad de lectura/escritura, elija Editar.

   c. En la sección Modo de capacidad, elija Aprovisionado.

   d. En la sección Capacidad de tabla, establezca Escalado automático en Activado para Capacidad de lectura, Capacidad de escritura o ambas. Para cada uno de ellos, establezca la política de escalado que desee para la tabla y, opcionalmente, para todos los índices secundarios globales de la tabla.

      - Unidades de capacidad mínimas: introduzca el límite inferior del intervalo de escalamiento automático.
      - Unidades de capacidad máxima: introduzca el límite superior del intervalo de escalamiento automático.
      - Objetivo de utilización: introduzca su porcentaje de utilización objetivo para la tabla.
      - Usar la misma configuración de capacidad de lectura/escritura para todos los índices secundarios globales: elija si los índices secundarios globales deben utilizar la misma política de escalamiento automático que la tabla base.

      > **Note**
      >
      > Para obtener el máximo rendimiento, le recomendamos que habilite la opción Use the same read/write capacity settings for all global secondary indexes (Utilizar la misma configuración de capacidad de lectura/escritura para todos los índices secundarios globales). Esta opción permite que la función Auto Scaling de DynamoDB pueda escalar uniformemente todos los índices secundarios globales

de la tabla base. Esto incluye los índices secundarios globales existentes y cualquier otro que se creen en esta tabla en el futuro.
Con esta opción habilitada, no se puede establecer una política de escalado para un índice secundario global individual.

4. Cuando la configuración sea la que desea, elija Save (Guardar).

Visualización de las actividades de Auto Scaling en la consola

A medida que la aplicación envía tráfico de lectura y escritura a la tabla, la función Auto Scaling de DynamoDB modifica de forma dinámica la configuración de rendimiento de esa tabla. Amazon CloudWatch realiza un seguimiento de la capacidad aprovisionada y consumida, los eventos limitados, la latencia y otras métricas de todas las tablas de DynamoDB e índices secundarios.

Para ver estas métricas en la consola de DynamoDB, elija la tabla con la que desee trabajar y seleccione la pestaña Monitorear. Para crear una vista personalizable de las métricas de tabla, seleccione View all in CloudWatch (Ver todo en CloudWatch).

Modificación o deshabilitación de la configuración de Auto Scaling de DynamoDB

Puede utilizar la Consola de administración de AWS para modificar la configuración de Auto Scaling de DynamoDB. Para ello, vaya a la pestaña Configuración adicional de la tabla y elija Editar en la sección Capacidad de lectura/escritura. Para obtener más información sobre estas opciones, consulte Habilitación de la función Auto Scaling de DynamoDB en tablas existentes.

Uso de la AWS CLI para administrar la función Auto Scaling de DynamoDB

En lugar de utilizar la Consola de administración de AWS, puede utilizar la AWS Command Line Interface (AWS CLI) para administrar la función Auto Scaling de Amazon DynamoDB. En el tutorial de esta sección se muestra cómo instalar y configurar la AWS CLI para administrar la función Auto Scaling de DynamoDB. En este tutorial, aprenderá a hacer lo siguiente:

- Creación de una tabla de DynamoDB llamada `TestTable`. Los ajustes de desempeño iniciales son 5 unidades de capacidad de lectura y 5 unidades de capacidad de escritura.
- Cree una política Auto Scaling de aplicaciones para `TestTable`. La política está dirigida a mantener una proporción objetivo del 50 % entre la capacidad de escritura consumida y la capacidad de escritura provisionada. El rango de esta métrica está comprendido entre 5 y 10 unidades de capacidad de escritura. (Auto Scaling de aplicaciones no puede ajustar el rendimiento fuera de este rango).

- Ejecute un programa en Python para dirigir tráfico de escritura a `TestTable`. Cuando la proporción objetivo supere el 50 % durante un periodo prolongado, el Auto Scaling de aplicaciones se lo notificará a DynamoDB para que ajuste el rendimiento de `TestTable` al alza y, de este modo, mantener el 50 % de utilización de destinos.
- Compruebe que DynamoDB haya ajustado correctamente la capacidad de escritura aprovisionada para `TestTable`.

> **Note**
>
> También puede programar el escalado de DynamoDB para que se realice en determinados momentos. Descubra los pasos básicos aquí.

Temas

- Antes de empezar
- Paso 1: crear una tabla de DynamoDB
- Paso 2: registrar un objetivo escalable
- Paso 3: crear una política de escalado
- Paso 4: dirigir tráfico de escritura a TestTable
- Paso 5: consultar las acciones de Auto Scaling de aplicaciones
- (Opcional) Paso 6: limpiar

Antes de empezar

Complete las siguientes tareas antes de comenzar el tutorial.

Instala la AWS CLI

Si aún no lo ha hecho, debe instalar y configurar la AWS CLI. Para ello, siga las siguientes instrucciones en la Guía del usuario de AWS Command Line Interface:

- Instalación del AWS CLI
- Configuración de la AWS CLI

Instalación de Python

Una parte de este tutorial requiere que se ejecute un programa en Python (consulte Paso 4: dirigir tráfico de escritura a TestTable). Si aún no lo tiene instalado, puede descargar Python.

Paso 1: crear una tabla de DynamoDB

En este paso, utilice la AWS CLI para crear una `TestTable`. La clave principal consta de `pk` (clave de partición) y `sk` (clave de ordenación). Ambos atributos son de tipo `Number`. Los ajustes de desempeño iniciales son 5 unidades de capacidad de lectura y 5 unidades de capacidad de escritura.

1. Utilice el siguiente comando de la AWS CLI para crear la tabla.

```
aws dynamodb create-table \
    --table-name TestTable \
    --attribute-definitions \
        AttributeName=pk,AttributeType=N \
        AttributeName=sk,AttributeType=N \
    --key-schema \
        AttributeName=pk,KeyType=HASH \
        AttributeName=sk,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
```

2. Para comprobar el estado de la tabla, use el comando siguiente.

```
aws dynamodb describe-table \
    --table-name TestTable \
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

   La tabla está lista para usarla cuando su estado es `ACTIVE`.

Paso 2: registrar un objetivo escalable

Ahora, vamos a registrar la capacidad de escritura de la tabla como objetivo escalable con Auto Scaling de aplicaciones. Esto permite que Auto Scaling de aplicaciones ajuste la capacidad de escritura aprovisionada para TestTable, pero solo dentro del rango de entre 5 y 10 unidades de capacidad.

> **Note**
>
> La función Auto Scaling de DynamoDB requiere la presencia de un rol (`AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`) que lleve a cabo las acciones de escalado automático en su nombre. Este rol se crea automáticamente para usted. Para obtener más información, consulte Roles vinculados a servicios de Application Auto Scaling en la Guía del usuario de Application Auto Scaling.

1. Ingrese el siguiente comando para registrar el objetivo escalable.

```
aws application-autoscaling register-scalable-target \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --min-capacity 5 \
    --max-capacity 10
```

2. Para comprobar el registro, utilice el siguiente comando.

```
aws application-autoscaling describe-scalable-targets \
    --service-namespace dynamodb \
    --resource-id "table/TestTable"
```

> **Note**
>
> También puede registrar un destino escalable en un índice secundario global. Por ejemplo, para un índice secundario global (“test-index”), el ID de recurso y los argumentos de dimensión escalable se actualizan adecuadamente.
>
> ```
> aws application-autoscaling register-scalable-target \
>     --service-namespace dynamodb \
>     --resource-id "table/TestTable/index/test-index" \
>     --scalable-dimension "dynamodb:index:WriteCapacityUnits" \
>     --min-capacity 5 \
>     --max-capacity 10
> ```

Paso 3: crear una política de escalado

En este paso, se crea una política de escalado para `TestTable`. La política define los detalles según los cuales Auto Scaling de aplicaciones puede ajustar el rendimiento aprovisionado de la tabla y las acciones llevará a cabo para ello. Puede asociar esta política al objetivo escalable definido en el paso anterior (unidades de capacidad de escritura para la tabla `TestTable`).

La política contiene los componentes siguientes:

- `PredefinedMetricSpecification`: métrica que puede ajustar Auto Scaling de aplicaciones. Para DynamoDB, los siguientes valores son válidos para `PredefinedMetricType`:
  - `DynamoDBReadCapacityUtilization`
  - `DynamoDBWriteCapacityUtilization`
- `ScaleOutCooldown`: cantidad mínima de tiempo (en segundos) entre cada evento de Auto Scaling de aplicaciones que aumenta el rendimiento aprovisionado. Este parámetro permite que Auto Scaling de aplicaciones aumente de forma continua, pero no drástica, el rendimiento en respuesta a las cargas de trabajo reales. El ajuste predeterminado de `ScaleOutCooldown` es 0.
- `ScaleInCooldown`: cantidad mínima de tiempo (en segundos) entre cada evento de Auto Scaling de aplicaciones que reduce el rendimiento aprovisionado. Este parámetro permite que Auto Scaling de aplicaciones disminuya el rendimiento de manera gradual y predecible. El ajuste predeterminado de `ScaleInCooldown` es 0.
- `TargetValue`: Auto Scaling de aplicaciones se asegura de que la proporción entre capacidad consumida y capacidad aprovisionada se mantenga en este valor o en un valor próximo. `TargetValue` se define como un porcentaje.

> **Note**
>
> Para entender cómo funciona `TargetValue`, imagine que tiene una tabla con una configuración de rendimiento aprovisionado de 200 unidades de capacidad de escritura. Decide crear una política de escalado para esta tabla, con un valor de `TargetValue` del 70 %.
>
> Ahora, supongamos que comienza a dirigir el tráfico de escritura a la tabla, de tal forma que el rendimiento de escritura real es de 150 unidades de capacidad. La proporción entre capacidad consumida y aprovisionada es ahora de (150/200), es decir, del 75 %. Esta proporción supera su objetivo, de modo que Auto Scaling de aplicaciones aumenta la capacidad de escritura aprovisionada a 215 para que la proporción sea de (150/215),

es decir, del 69,77 %; de esta forma se mantiene lo más próxima posible al valor de `TargetValue`, pero sin superarlo.

Para `TestTable`, configure `TargetValue` hasta el 50 %. Auto Scaling de aplicaciones ajusta el rendimiento aprovisionado de la tabla dentro del rango comprendido entre 5 y 10 unidades de capacidad (consulte Paso 2: registrar un objetivo escalable), de tal forma que la proporción entre capacidad consumida y provisionada se mantiene en el 50 % o en un valor próximo a este. Los valores de `ScaleOutCooldown` y `ScaleInCooldown` se establecen en 60 segundos.

1. Cree un archivo denominado `scaling-policy.json` con el siguiente contenido.

```
{
    "PredefinedMetricSpecification": {
        "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
    },
    "ScaleOutCooldown": 60,
    "ScaleInCooldown": 60,
    "TargetValue": 50.0
}
```

2. Utilice el siguiente comando de la AWS CLI para crear la política.

```
aws application-autoscaling put-scaling-policy \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --policy-name "MyScalingPolicy" \
    --policy-type "TargetTrackingScaling" \
    --target-tracking-scaling-policy-configuration file://scaling-policy.json
```

3. En el resultado, observe que Auto Scaling de aplicaciones ha creado dos alarmas de Amazon CloudWatch, una para cada límite (superior e inferior) del rango de escalado objetivo.
4. Utilice el comando de AWS CLI siguiente para ver más detalles sobre la política de escalado.

```
aws application-autoscaling describe-scaling-policies \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --policy-name "MyScalingPolicy"
```

5. En el resultado, compruebe que los ajustes de la política coincidan con las especificaciones de Paso 2: registrar un objetivo escalable y Paso 3: crear una política de escalado.

Paso 4: dirigir tráfico de escritura a TestTable

Ahora puede probar la política de escalado escribiendo datos en . `TestTable`. Para ello, deberá ejecutar un programa en Python.

1. Cree un archivo denominado `bulk-load-test-table.py` con el siguiente contenido.

```
import boto3
dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table("TestTable")

filler = "x" * 100000

i = 0
while (i < 10):
    j = 0
    while (j < 10):
        print (i, j)

        table.put_item(
            Item={
                'pk':i,
                'sk':j,
                'filler':{"S":filler}
            }
        )
        j += 1
    i += 1
```

2. Para ejecutar el programa, introduzca el siguiente comando.

   ```
   python bulk-load-test-table.py
   ```

   La capacidad de escritura provisionada de `TestTable` es muy baja (5 unidades de capacidad de escritura), por lo que el programa se ahoga en ocasiones a causa de la limitación controlada de escritura. Este es el comportamiento esperado.

   Permita que el programa continúe ejecutándose mientras avanza al paso siguiente.

Paso 5: consultar las acciones de Auto Scaling de aplicaciones

En este paso, consultaremos las acciones de Auto Scaling de aplicaciones que se han iniciado en su nombre. Además, comprobaremos que Auto Scaling de aplicaciones ha actualizado la capacidad de escritura provisionada para `TestTable`.

1. Ingrese el siguiente comando para ver las acciones de Auto Scaling de aplicaciones.

   ```
   aws application-autoscaling describe-scaling-activities \
       --service-namespace dynamodb
   ```

   Vuelva a ejecutar este comando cada cierto tiempo mientras el programa en Python siga en ejecución. (Se tardarán varios minutos hasta que se invoque la política de escalado). En algún momento, debería aparecer el resultado siguiente.

   ```
   ...
   {
       "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
       "Description": "Setting write capacity units to 10.",
       "ResourceId": "table/TestTable",
       "ActivityId": "0cc6fb03-2a7c-4b51-b67f-217224c6b656",
       "StartTime": 1489088210.175,
       "ServiceNamespace": "dynamodb",
       "EndTime": 1489088246.85,
       "Cause": "monitor alarm AutoScaling-table/TestTable-
   AlarmHigh-1bb3c8db-1b97-4353-baf1-4def76f4e1b9 in state ALARM triggered policy
    MyScalingPolicy",
       "StatusMessage": "Successfully set write capacity units to 10. Change
    successfully fulfilled by dynamodb.",
       "StatusCode": "Successful"
   },
   ...
   ```

   Esto indica que el Auto Scaling de aplicaciones ha emitido una solicitud `UpdateTable` a DynamoDB.

2. Ingrese el siguiente comando para comprobar que DynamoDB ha aumentado la capacidad de escritura de la tabla.

   ```
   aws dynamodb describe-table \
       --table-name TestTable \
   ```

```
    --query "Table.[TableName,TableStatus,ProvisionedThroughput]"
```

`WriteCapacityUnits` debería haberse escalado de `5` a `10`.

(Opcional) Paso 6: limpiar

En este tutorial, ha creado varios recursos. Puede eliminar estos recursos cuando ya los necesite.

1. Elimine la política de escalado para `TestTable`.

```
aws application-autoscaling delete-scaling-policy \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits" \
    --policy-name "MyScalingPolicy"
```

2. Anule el registro del objetivo escalable.

```
aws application-autoscaling deregister-scalable-target \
    --service-namespace dynamodb \
    --resource-id "table/TestTable" \
    --scalable-dimension "dynamodb:table:WriteCapacityUnits"
```

3. Elimine la tabla `TestTable`.

```
aws dynamodb delete-table --table-name TestTable
```

Uso del AWS SDK para configurar escalado automático en tablas de Amazon DynamoDB

Además de utilizar la Consola de administración de AWS y la AWS Command Line Interface (AWS CLI), puede escribir aplicaciones que interaccionen con el escalado automático de Amazon DynamoDB. Esta sección contiene dos programas de Java que puede utilizar para probar esta funcionalidad:

- `EnableDynamoDBAutoscaling.java`
- `DisableDynamoDBAutoscaling.java`

Habilitación de Auto Scaling de aplicaciones para una tabla

En el siguiente programa se muestra un ejemplo de cómo configurar una política de escalado automático para una tabla de DynamoDB (`TestTable`). Funciona de la siguiente manera:

- El programa registra las unidades de capacidad de escritura como objetivo escalable de `TestTable`. El rango de esta métrica está comprendido entre 5 y 10 unidades de capacidad de escritura.
- Después de crear el objetivo escalable, el programa crea una configuración de seguimiento del objetivo. La política está dirigida a mantener una proporción objetivo del 50 % entre la capacidad de escritura consumida y la capacidad de escritura provisionada.
- Después, el programa crea la política de escalado basada en la configuración de seguimiento del objetivo.

> **Note**
>
> Cuando se elimina manualmente una tabla o réplica de tabla global, no se eliminan automáticamente los destinos escalables asociados, las políticas de escalado o las alarmas de CloudWatch.

Java v2

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.applicationautoscaling.ApplicationAutoScalingClient;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ApplicationAutoScalingExceptio
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRespons
import software.amazon.awssdk.services.applicationautoscaling.model.PolicyType;
import
 software.amazon.awssdk.services.applicationautoscaling.model.PredefinedMetricSpecification;
```

```
import
 software.amazon.awssdk.services.applicationautoscaling.model.PutScalingPolicyRequest;
import
 software.amazon.awssdk.services.applicationautoscaling.model.RegisterScalableTargetRequest;
import software.amazon.awssdk.services.applicationautoscaling.model.ScalingPolicy;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ServiceNamespace;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ScalableDimension;
import software.amazon.awssdk.services.applicationautoscaling.model.MetricType;
import
 software.amazon.awssdk.services.applicationautoscaling.model.TargetTrackingScalingPolicyCon
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class EnableDynamoDBAutoscaling {
    public static void main(String[] args) {
        final String usage = """

            Usage:
               <tableId> <roleARN> <policyName>\s

            Where:
               tableId - The table Id value (for example, table/Music).
               roleARN - The ARN of the role that has ApplicationAutoScaling
 permissions.
               policyName - The name of the policy to create.

            """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        System.out.println("This example registers an Amazon DynamoDB table, which
 is the resource to scale.");
```

```
        String tableId = args[0];
        String roleARN = args[1];
        String policyName = args[2];
        ServiceNamespace ns = ServiceNamespace.DYNAMODB;
        ScalableDimension tableWCUs =
ScalableDimension.DYNAMODB_TABLE_WRITE_CAPACITY_UNITS;
        ApplicationAutoScalingClient appAutoScalingClient =
ApplicationAutoScalingClient.builder()
            .region(Region.US_EAST_1)
            .build();

        registerScalableTarget(appAutoScalingClient, tableId, roleARN, ns,
tableWCUs);
        verifyTarget(appAutoScalingClient, tableId, ns, tableWCUs);
        configureScalingPolicy(appAutoScalingClient, tableId, ns, tableWCUs,
policyName);
    }

    public static void registerScalableTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, String roleARN, ServiceNamespace ns,
ScalableDimension tableWCUs) {
        try {
            RegisterScalableTargetRequest targetRequest =
RegisterScalableTargetRequest.builder()
                .serviceNamespace(ns)
                .scalableDimension(tableWCUs)
                .resourceId(tableId)
                .roleARN(roleARN)
                .minCapacity(5)
                .maxCapacity(10)
                .build();

            appAutoScalingClient.registerScalableTarget(targetRequest);
            System.out.println("You have registered " + tableId);

        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    // Verify that the target was created.
    public static void verifyTarget(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
```

```
        DescribeScalableTargetsRequest dscRequest =
DescribeScalableTargetsRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceIds(tableId)
            .build();

        DescribeScalableTargetsResponse response =
appAutoScalingClient.describeScalableTargets(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }

    // Configure a scaling policy.
    public static void configureScalingPolicy(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs, String policyName) {
        // Check if the policy exists before creating a new one.
        DescribeScalingPoliciesResponse describeScalingPoliciesResponse =
appAutoScalingClient.describeScalingPolicies(DescribeScalingPoliciesRequest.builder()
            .serviceNamespace(ns)
            .resourceId(tableId)
            .scalableDimension(tableWCUs)
            .build());

        if (!describeScalingPoliciesResponse.scalingPolicies().isEmpty()) {
            // If policies exist, consider updating an existing policy instead of
creating a new one.
            System.out.println("Policy already exists. Consider updating it
instead.");
            List<ScalingPolicy> polList =
describeScalingPoliciesResponse.scalingPolicies();
            for (ScalingPolicy pol : polList) {
                System.out.println("Policy name:" +pol.policyName());
            }
        } else {
            // If no policies exist, proceed with creating a new policy.
            PredefinedMetricSpecification specification =
PredefinedMetricSpecification.builder()

.predefinedMetricType(MetricType.DYNAMO_DB_WRITE_CAPACITY_UTILIZATION)
                .build();
```

```
            TargetTrackingScalingPolicyConfiguration policyConfiguration =
 TargetTrackingScalingPolicyConfiguration.builder()
                .predefinedMetricSpecification(specification)
                .targetValue(50.0)
                .scaleInCooldown(60)
                .scaleOutCooldown(60)
                .build();

            PutScalingPolicyRequest putScalingPolicyRequest =
 PutScalingPolicyRequest.builder()
                .targetTrackingScalingPolicyConfiguration(policyConfiguration)
                .serviceNamespace(ns)
                .scalableDimension(tableWCUs)
                .resourceId(tableId)
                .policyName(policyName)
                .policyType(PolicyType.TARGET_TRACKING_SCALING)
                .build();

            try {
                appAutoScalingClient.putScalingPolicy(putScalingPolicyRequest);
                System.out.println("You have successfully created a scaling policy
 for an Application Auto Scaling scalable target");
            } catch (ApplicationAutoScalingException e) {
                System.err.println("Error: " + e.awsErrorDetails().errorMessage());
            }
        }
    }
}
```

Java v1

El programa requiere que se suministre el nombre de recurso de Amazon (ARN) de un rol vinculado a un servicio de Auto Scaling de aplicaciones válido. (Por ejemplo: `arn:aws:iam::122517410325:role/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`.) En el siguiente programa, sustituya `SERVICE_ROLE_ARN_GOES_HERE` por el ARN real.

```
package com.amazonaws.codesamples.autoscaling;

import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClient;
import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClientBuilder;
```

```
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsResult;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesResult;
import com.amazonaws.services.applicationautoscaling.model.MetricType;
import com.amazonaws.services.applicationautoscaling.model.PolicyType;
import
 com.amazonaws.services.applicationautoscaling.model.PredefinedMetricSpecification;
import com.amazonaws.services.applicationautoscaling.model.PutScalingPolicyRequest;
import
 com.amazonaws.services.applicationautoscaling.model.RegisterScalableTargetRequest;
import com.amazonaws.services.applicationautoscaling.model.ScalableDimension;
import com.amazonaws.services.applicationautoscaling.model.ServiceNamespace;
import
 com.amazonaws.services.applicationautoscaling.model.TargetTrackingScalingPolicyConfiguratio

public class EnableDynamoDBAutoscaling {

 static AWSApplicationAutoScalingClient aaClient = (AWSApplicationAutoScalingClient)
 AWSApplicationAutoScalingClientBuilder
   .standard().build();

 public static void main(String args[]) {

  ServiceNamespace ns = ServiceNamespace.Dynamodb;
  ScalableDimension tableWCUs = ScalableDimension.DynamodbTableWriteCapacityUnits;
  String resourceID = "table/TestTable";

  // Define the scalable target
  RegisterScalableTargetRequest rstRequest = new RegisterScalableTargetRequest()
    .withServiceNamespace(ns)
    .withResourceId(resourceID)
    .withScalableDimension(tableWCUs)
    .withMinCapacity(5)
    .withMaxCapacity(10)
    .withRoleARN("SERVICE_ROLE_ARN_GOES_HERE");

  try {
   aaClient.registerScalableTarget(rstRequest);
  } catch (Exception e) {
```

```
  System.err.println("Unable to register scalable target: ");
  System.err.println(e.getMessage());
 }

 // Verify that the target was created
 DescribeScalableTargetsRequest dscRequest = new DescribeScalableTargetsRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceIds(resourceID);
 try {
  DescribeScalableTargetsResult dsaResult =
aaClient.describeScalableTargets(dscRequest);
  System.out.println("DescribeScalableTargets result: ");
  System.out.println(dsaResult);
  System.out.println();
 } catch (Exception e) {
  System.err.println("Unable to describe scalable target: ");
  System.err.println(e.getMessage());
 }

 System.out.println();

 // Configure a scaling policy
 TargetTrackingScalingPolicyConfiguration targetTrackingScalingPolicyConfiguration
= new TargetTrackingScalingPolicyConfiguration()
   .withPredefinedMetricSpecification(
     new PredefinedMetricSpecification()
       .withPredefinedMetricType(MetricType.DynamoDBWriteCapacityUtilization))
   .withTargetValue(50.0)
   .withScaleInCooldown(60)
   .withScaleOutCooldown(60);

 // Create the scaling policy, based on your configuration
 PutScalingPolicyRequest pspRequest = new PutScalingPolicyRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID)
   .withPolicyName("MyScalingPolicy")
   .withPolicyType(PolicyType.TargetTrackingScaling)

.withTargetTrackingScalingPolicyConfiguration(targetTrackingScalingPolicyConfiguration);

 try {
  aaClient.putScalingPolicy(pspRequest);
```

```
  } catch (Exception e) {
   System.err.println("Unable to put scaling policy: ");
   System.err.println(e.getMessage());
  }

  // Verify that the scaling policy was created
  DescribeScalingPoliciesRequest dspRequest = new DescribeScalingPoliciesRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceId(resourceID);

  try {
   DescribeScalingPoliciesResult dspResult =
 aaClient.describeScalingPolicies(dspRequest);
   System.out.println("DescribeScalingPolicies result: ");
   System.out.println(dspResult);
  } catch (Exception e) {
   e.printStackTrace();
   System.err.println("Unable to describe scaling policy: ");
   System.err.println(e.getMessage());
  }

 }

}
```

Desactivación de Auto Scaling de aplicaciones para una tabla

En el siguiente programa se invierte el proceso anterior. Se elimina la política de escalado automático y, a continuación, se anula el registro del objetivo escalable.

Java v2

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.applicationautoscaling.ApplicationAutoScalingClient;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ApplicationAutoScalingExceptio
import
 software.amazon.awssdk.services.applicationautoscaling.model.DeleteScalingPolicyRequest;
```

```
import
 software.amazon.awssdk.services.applicationautoscaling.model.DeregisterScalableTargetReques
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalableTargetsRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRequest
import
 software.amazon.awssdk.services.applicationautoscaling.model.DescribeScalingPoliciesRespons
import
 software.amazon.awssdk.services.applicationautoscaling.model.ScalableDimension;
import
 software.amazon.awssdk.services.applicationautoscaling.model.ServiceNamespace;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class DisableDynamoDBAutoscaling {
    public static void main(String[] args) {
        final String usage = """

            Usage:
               <tableId> <policyName>\s

            Where:
               tableId - The table Id value (for example, table/Music).\s
               policyName - The name of the policy (for example, $Music5-scaling-
policy).

            """;
        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        ApplicationAutoScalingClient appAutoScalingClient =
 ApplicationAutoScalingClient.builder()
```

```
            .region(Region.US_EAST_1)
            .build();

        ServiceNamespace ns = ServiceNamespace.DYNAMODB;
        ScalableDimension tableWCUs =
ScalableDimension.DYNAMODB_TABLE_WRITE_CAPACITY_UNITS;
        String tableId = args[0];
        String policyName = args[1];

        deletePolicy(appAutoScalingClient, policyName, tableWCUs, ns, tableId);
        verifyScalingPolicies(appAutoScalingClient, tableId, ns, tableWCUs);
        deregisterScalableTarget(appAutoScalingClient, tableId, ns, tableWCUs);
        verifyTarget(appAutoScalingClient, tableId, ns, tableWCUs);
    }

    public static void deletePolicy(ApplicationAutoScalingClient
appAutoScalingClient, String policyName, ScalableDimension tableWCUs,
ServiceNamespace ns, String tableId) {
        try {
            DeleteScalingPolicyRequest delSPRequest =
DeleteScalingPolicyRequest.builder()
                .policyName(policyName)
                .scalableDimension(tableWCUs)
                .serviceNamespace(ns)
                .resourceId(tableId)
                .build();

            appAutoScalingClient.deleteScalingPolicy(delSPRequest);
            System.out.println(policyName +" was deleted successfully.");

        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    // Verify that the scaling policy was deleted
    public static void verifyScalingPolicies(ApplicationAutoScalingClient
appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
tableWCUs) {
        DescribeScalingPoliciesRequest dscRequest =
DescribeScalingPoliciesRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceId(tableId)
```

```
            .build();

        DescribeScalingPoliciesResponse response =
 appAutoScalingClient.describeScalingPolicies(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }

    public static void deregisterScalableTarget(ApplicationAutoScalingClient
 appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
 tableWCUs) {
        try {
            DeregisterScalableTargetRequest targetRequest =
 DeregisterScalableTargetRequest.builder()
                .scalableDimension(tableWCUs)
                .serviceNamespace(ns)
                .resourceId(tableId)
                .build();

            appAutoScalingClient.deregisterScalableTarget(targetRequest);
            System.out.println("The scalable target was deregistered.");

        } catch (ApplicationAutoScalingException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }

    public static void verifyTarget(ApplicationAutoScalingClient
 appAutoScalingClient, String tableId, ServiceNamespace ns, ScalableDimension
 tableWCUs) {
        DescribeScalableTargetsRequest dscRequest =
 DescribeScalableTargetsRequest.builder()
            .scalableDimension(tableWCUs)
            .serviceNamespace(ns)
            .resourceIds(tableId)
            .build();

        DescribeScalableTargetsResponse response =
 appAutoScalingClient.describeScalableTargets(dscRequest);
        System.out.println("DescribeScalableTargets result: ");
        System.out.println(response);
    }
}
```

Java v1

```
package com.amazonaws.codesamples.autoscaling;

import
 com.amazonaws.services.applicationautoscaling.AWSApplicationAutoScalingClient;
import
 com.amazonaws.services.applicationautoscaling.model.DeleteScalingPolicyRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DeregisterScalableTargetRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalableTargetsResult;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesRequest;
import
 com.amazonaws.services.applicationautoscaling.model.DescribeScalingPoliciesResult;
import com.amazonaws.services.applicationautoscaling.model.ScalableDimension;
import com.amazonaws.services.applicationautoscaling.model.ServiceNamespace;

public class DisableDynamoDBAutoscaling {

 static AWSApplicationAutoScalingClient aaClient = new
 AWSApplicationAutoScalingClient();

 public static void main(String args[]) {

  ServiceNamespace ns = ServiceNamespace.Dynamodb;
  ScalableDimension tableWCUs = ScalableDimension.DynamodbTableWriteCapacityUnits;
  String resourceID = "table/TestTable";

  // Delete the scaling policy
  DeleteScalingPolicyRequest delSPRequest = new DeleteScalingPolicyRequest()
    .withServiceNamespace(ns)
    .withScalableDimension(tableWCUs)
    .withResourceId(resourceID)
    .withPolicyName("MyScalingPolicy");

  try {
   aaClient.deleteScalingPolicy(delSPRequest);
  } catch (Exception e) {
   System.err.println("Unable to delete scaling policy: ");
   System.err.println(e.getMessage());
```

```
 }

 // Verify that the scaling policy was deleted
 DescribeScalingPoliciesRequest descSPRequest = new
DescribeScalingPoliciesRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID);

 try {
  DescribeScalingPoliciesResult dspResult =
aaClient.describeScalingPolicies(descSPRequest);
  System.out.println("DescribeScalingPolicies result: ");
  System.out.println(dspResult);
 } catch (Exception e) {
  e.printStackTrace();
  System.err.println("Unable to describe scaling policy: ");
  System.err.println(e.getMessage());
 }

 System.out.println();

 // Remove the scalable target
 DeregisterScalableTargetRequest delSTRequest = new
DeregisterScalableTargetRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceId(resourceID);

 try {
  aaClient.deregisterScalableTarget(delSTRequest);
 } catch (Exception e) {
  System.err.println("Unable to deregister scalable target: ");
  System.err.println(e.getMessage());
 }

 // Verify that the scalable target was removed
 DescribeScalableTargetsRequest dscRequest = new DescribeScalableTargetsRequest()
   .withServiceNamespace(ns)
   .withScalableDimension(tableWCUs)
   .withResourceIds(resourceID);

 try {
```

```
    DescribeScalableTargetsResult dsaResult =
  aaClient.describeScalableTargets(dscRequest);
    System.out.println("DescribeScalableTargets result: ");
    System.out.println(dsaResult);
    System.out.println();
   } catch (Exception e) {
    System.err.println("Unable to describe scalable target: ");
    System.err.println(e.getMessage());
   }

  }

}
```

Capacidad reservada de DynamoDB

En el caso de las tablas de capacidad aprovisionada que utilizan la clase de tabla estándar, DynamoDB ofrece la posibilidad de comprar capacidad reservada para la capacidad de lectura y escritura. La compra de capacidad reservada es un acuerdo por el que se paga una cantidad mínima de capacidad de rendimiento aprovisionada durante el período de vigencia del acuerdo, a cambio de descuentos en los precios.

> **Note**
>
> No puede adquirir capacidad reservada para las unidades de capacidad de escritura replicadas (rWCU). La capacidad reservada solo se aplica a la región en la que se compró. La capacidad reservada tampoco está disponible para tablas que utilizan la clase de tabla DynamoDB Standard - IA o el modo de capacidad bajo demanda.

La capacidad reservada se adquiere en asignaciones de 100 WCU o 100 RCU. La oferta de capacidad reservada más pequeña es de 100 unidades de capacidad (lectura o escritura). La capacidad reservada de DynamoDB se ofrece con un compromiso de un año o, en determinadas regiones, de tres años. Puede disfrutar de un ahorro de hasta el 54 % sobre las tarifas estándar con el compromiso de un año y de un 77 % sobre las tarifas estándar en el de tres años. Para obtener más información sobre cómo y cuándo debería adquirirlos, consulte Amazon DynamoDB Reserved Capacity.

Cuando se adquiere capacidad reservada de DynamoDB, realiza un único pago parcial por adelantado y recibe una tarifa por hora con descuento por el uso aprovisionado comprometido. Se paga por todo el uso aprovisionado comprometido, independientemente del uso real, por lo que el ahorro de costos depende enormemente del uso. Cualquier capacidad que aprovisione que supere la capacidad reservada adquirida se cobrará a la tarifa de capacidad aprovisionada estándar. Al reservar las unidades de capacidad de lectura y escritura por adelantado, logrará un ahorro importante en los costos de capacidad aprovisionada.

No puede vender, cancelar ni transferir la capacidad reservada a otra región o cuenta.

Descripción del rendimiento en caliente de DynamoDB

El rendimiento en caliente se refiere al número de operaciones de lectura y escritura que la tabla de DynamoDB puede admitir de modo instantáneo. Estos valores están disponibles de manera predeterminada para todas las tablas y los índices secundarios globales (GSI) y representan cuánto han escalado en función del uso histórico. Si utiliza el modo bajo demanda, o si actualiza el rendimiento aprovisionado a estos valores, la aplicación podrá emitir solicitudes hasta esos valores de forma instantánea.

DynamoDB ajustará automáticamente los valores de rendimiento en caliente a medida que aumente el uso. No obstante, también puede aumentar estos valores de forma proactiva cuando sea necesario, lo que resulta especialmente útil en los picos de eventos próximos, como lanzamientos de productos o ventas. En el caso de eventos de pico planificados, en los que las tasas de solicitudes para la tabla de DynamoDB podrían aumentar 10 veces, 100 veces o más, ahora puede evaluar si el rendimiento en caliente actual es suficiente para gestionar el tráfico previsto. Si no es así, puede aumentar el valor de rendimiento en caliente sin cambiar la configuración de rendimiento ni el modo de facturación. Este proceso se denomina precalentamiento de una tabla, lo que le permite establecer una línea de base que las tablas pueden admitir al instante. Esto garantiza que las aplicaciones puedan manejar tasas de solicitudes más elevadas desde el momento en que se produzcan.

Puede aumentar el valor de rendimiento en caliente en el caso de las operaciones de lectura, escritura o ambas. Puede aumentar este valor para las tablas de una sola región, las tablas globales y los GSI nuevos y existentes. En el caso de las tablas globales, esta característica está disponible para la versión 2019.11.21 (actual) y la configuración de rendimiento en caliente que establezca se aplicará de forma automática a todas las tablas de réplica de la tabla global. No hay límite en el número de tablas de DynamoDB a las que puede aplicar el precalentamiento en cualquier momento. El tiempo necesario para completar el precalentamiento depende de los valores que establezca y

del tamaño de la tabla o el índice. Puede presentar solicitudes de precalentamiento simultáneas y no interferirán en ninguna operación de tabla. Puede precalentar la tabla hasta el límite de cuota de tabla o índice de la cuenta en esa región. Utilice la consola de Service Quotas para comprobar los límites actuales y aumentarlos si es necesario.

Los valores de rendimiento en caliente están disponibles de forma predeterminada para todas las tablas y los índices secundarios sin costo alguno. No obstante, si aumenta de forma proactiva estos valores de rendimiento en caliente predeterminados para precalentar las tablas, se le cobrarán esas solicitudes. Para obtener más información, consulte los precios de Amazon DynamoDB.

Para obtener más información sobre el rendimiento en caliente, consulte los temas siguientes:

Temas

- Comprobación del rendimiento en caliente actual de la tabla de DynamoDB
- Aumento del rendimiento en caliente de la tabla de DynamoDB existente
- Creación de una tabla de DynamoDB nueva con rendimiento en caliente más alto
- Descripción del rendimiento en caliente de DynamoDB en diferentes escenarios

Comprobación del rendimiento en caliente actual de la tabla de DynamoDB

Utilice las siguientes instrucciones de la AWS CLI y de la consola de AWS para comprobar el valor actual de rendimiento en caliente de la tabla o el índice.

Consola de administración de AWS

Para comprobar el rendimiento en caliente de la tabla de DynamoDB con la consola de DynamoDB:

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación izquierdo, elija Tables (Tablas).
3. En la página Tablas, elija la tabla que desee.
4. Seleccione Configuración para ver el valor de rendimiento en caliente actual. Este valor se muestra como unidades de lectura por segundo y unidades de escritura por segundo.

AWS CLI

En el siguiente ejemplo de la AWS CLI se muestra cómo comprobar el rendimiento en caliente de la tabla de DynamoDB.

1. Ejecute la operación `describe-table` en la tabla de DynamoDB.

```
aws dynamodb describe-table --table-name GameScores
```

2. Verá una respuesta similar a la siguiente. La configuración de `WarmThroughput` se mostrará como `ReadUnitsPerSecond` y `WriteUnitsPerSecond`. El `Status` será `UPDATING` cuando se actualice el valor de rendimiento en caliente y `ACTIVE` cuando se establezca el nuevo valor de rendimiento en caliente.

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": 1726128388.729,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
```

```
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/GameScores",
        "TableId": "XXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST",
            "LastUpdateToPayPerRequestDateTime": 1726128388.729
        },
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 0,
                    "WriteCapacityUnits": 0
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/
GameScores/index/GameTitleIndex",
                "WarmThroughput": {
                    "ReadUnitsPerSecond": 12000,
                    "WriteUnitsPerSecond": 4000,
                    "Status": "ACTIVE"
                }
            }
        ],
```

```
        "DeletionProtectionEnabled": false,
        "WarmThroughput": {
            "ReadUnitsPerSecond": 12000,
            "WriteUnitsPerSecond": 4000,
            "Status": "ACTIVE"
        }
    }
}
```

Aumento del rendimiento en caliente de la tabla de DynamoDB existente

Una vez que haya comprobado el valor actual de rendimiento en caliente de la tabla de DynamoDB, puede actualizarlo con los siguientes pasos:

Consola de administración de AWS

Para comprobar el valor de rendimiento en caliente de la tabla de DynamoDB mediante la consola de DynamoDB:

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación izquierdo, elija Tables (Tablas).
3. En la página Tablas, elija la tabla que desee.
4. En el campo Rendimiento en caliente, seleccione Editar.
5. En la página Editar rendimiento en caliente, seleccione Aumentar el rendimiento en caliente.
6. Ajuste las unidades de lectura por segundo y las unidades de escritura por segundo. Estas dos configuraciones definen el rendimiento que la tabla puede gestionar al instante.
7. Seleccione Guardar.
8. Las unidades de lectura por segundo y las unidades de escritura por segundo se actualizarán en el campo Rendimiento en caliente cuando la solicitud termine de procesarse.

> **Note**
>
> La actualización del valor de rendimiento en caliente es una tarea asíncrona. El `Status` cambiará de `UPDATING` a `ACTIVE` cuando se complete la actualización.

AWS CLI

En el siguiente ejemplo de la AWS CLI se muestra cómo actualizar el valor de rendimiento en caliente de la tabla de DynamoDB.

1. Ejecute la operación `update-table` en la tabla de DynamoDB.

```
aws dynamodb update-table \
    --table-name GameScores \
    --warm-throughput ReadUnitsPerSecond=12345,WriteUnitsPerSecond=4567 \
    --global-secondary-index-updates \
        "[
            {
                \"Update\": {
                    \"IndexName\": \"GameTitleIndex\",
                    \"WarmThroughput\": {
                        \"ReadUnitsPerSecond\": 88,
                        \"WriteUnitsPerSecond\": 77
                    }
                }
            }
        ]" \
    --region us-east-1
```

2. Verá una respuesta similar a la siguiente. La configuración de `WarmThroughput` se mostrará como `ReadUnitsPerSecond` y `WriteUnitsPerSecond`. El `Status` será `UPDATING` cuando se actualice el valor de rendimiento en caliente y `ACTIVE` cuando se establezca el nuevo valor de rendimiento en caliente.

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
```

```
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": 1730242189.965,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 20,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/GameScores",
        "TableId": "XXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "ACTIVE",
```

```
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 50,
                    "WriteCapacityUnits": 25
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/
GameScores/index/GameTitleIndex",
                "WarmThroughput": {
                    "ReadUnitsPerSecond": 50,
                    "WriteUnitsPerSecond": 25,
                    "Status": "UPDATING"
                }
            }
        ],
        "DeletionProtectionEnabled": false,
        "WarmThroughput": {
            "ReadUnitsPerSecond": 12300,
            "WriteUnitsPerSecond": 4500,
            "Status": "UPDATING"
        }
    }
}
```

AWS SDK

En los siguientes ejemplos del SDK se muestra cómo actualizar el valor de rendimiento en caliente de la tabla de DynamoDB.

Java

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndexUpdate;
import
 software.amazon.awssdk.services.dynamodb.model.UpdateGlobalSecondaryIndexAction;
import software.amazon.awssdk.services.dynamodb.model.UpdateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;

...
public static WarmThroughput buildWarmThroughput(final Long readUnitsPerSecond,
```

```
                                                    final Long writeUnitsPerSecond) {
    return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
            .build();
}

public static void updateDynamoDBTable(DynamoDbClient ddb,
                                       String tableName,
                                       Long tableReadUnitsPerSecond,
                                       Long tableWriteUnitsPerSecond,
                                       String globalSecondaryIndexName,
                                       Long globalSecondaryIndexReadUnitsPerSecond,
                                       Long globalSecondaryIndexWriteUnitsPerSecond)
 {

    final WarmThroughput tableWarmThroughput =
 buildWarmThroughput(tableReadUnitsPerSecond, tableWriteUnitsPerSecond);
    final WarmThroughput gsiWarmThroughput =
 buildWarmThroughput(globalSecondaryIndexReadUnitsPerSecond,
 globalSecondaryIndexWriteUnitsPerSecond);

    final GlobalSecondaryIndexUpdate globalSecondaryIndexUpdate =
 GlobalSecondaryIndexUpdate.builder()
            .update(UpdateGlobalSecondaryIndexAction.builder()
                    .indexName(globalSecondaryIndexName)
                    .warmThroughput(gsiWarmThroughput)
                    .build()
            ).build();

    final UpdateTableRequest request = UpdateTableRequest.builder()
            .tableName(tableName)
            .globalSecondaryIndexUpdates(globalSecondaryIndexUpdate)
            .warmThroughput(tableWarmThroughput)
            .build();

    try {
        ddb.updateTable(request);
    } catch (DynamoDbException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }

    System.out.println("Done!");
```

```
}
```

Python

```
from boto3 import resource
from botocore.exceptions import ClientError

def update_dynamodb_table_warm_throughput(table_name, table_read_units,
 table_write_units, gsi_name, gsi_read_units, gsi_write_units, region_name="us-
east-1"):
    """
    Updates the warm throughput of a DynamoDB table and a global secondary index.

    :param table_name: The name of the table to update.
    :param table_read_units: The new read units per second for the table's warm
 throughput.
    :param table_write_units: The new write units per second for the table's warm
 throughput.
    :param gsi_name: The name of the global secondary index to update.
    :param gsi_read_units: The new read units per second for the GSI's warm
 throughput.
    :param gsi_write_units: The new write units per second for the GSI's warm
 throughput.
    :param region_name: The AWS Region name to target. defaults to us-east-1
    """
    try:
        ddb = resource('dynamodb', region_name)

        # Update the table's warm throughput
        table_warm_throughput = {
            "ReadUnitsPerSecond": table_read_units,
            "WriteUnitsPerSecond": table_write_units
        }

        # Update the global secondary index's warm throughput
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_read_units,
            "WriteUnitsPerSecond": gsi_write_units
        }

        # Construct the global secondary index update
        global_secondary_index_update = [
            {
```

```
                "Update": {
                    "IndexName": gsi_name,
                    "WarmThroughput": gsi_warm_throughput
                }
            }
        ]

        # Construct the update table request
        update_table_request = {
            "TableName": table_name,
            "GlobalSecondaryIndexUpdates": global_secondary_index_update,
            "WarmThroughput": table_warm_throughput
        }

        # Update the table
        ddb.update_table(**update_table_request)
        print("Table updated successfully!")
    except ClientError as e:
        print(f"Error updating table: {e}")
        raise e
```

Javascript

```
import { DynamoDBClient, UpdateTableCommand } from "@aws-sdk/client-dynamodb";

async function updateDynamoDBTableWarmThroughput(
  tableName,
  tableReadUnits,
  tableWriteUnits,
  gsiName,
  gsiReadUnits,
  gsiWriteUnits,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });

    // Construct the update table request
    const updateTableRequest = {
      TableName: tableName,
      GlobalSecondaryIndexUpdates: [
        {
            Update: {
```

```
                IndexName: gsiName,
                WarmThroughput: {
                    ReadUnitsPerSecond: gsiReadUnits,
                    WriteUnitsPerSecond: gsiWriteUnits,
                },
            },
        },
      ],
      WarmThroughput: {
          ReadUnitsPerSecond: tableReadUnits,
          WriteUnitsPerSecond: tableWriteUnits,
      },
    };

    const command = new UpdateTableCommand(updateTableRequest);
    const response = await ddbClient.send(command);
    console.log(`Table updated successfully! Response: ${response}`);
  } catch (error) {
    console.error(`Error updating table: ${error}`);
    throw error;
  }
}
```

Creación de una tabla de DynamoDB nueva con rendimiento en caliente más alto

Puede ajustar los valores de rendimiento en caliente cuando cree la tabla de DynamoDB según los pasos que se indican a continuación. Estos pasos también se aplican al crear una tabla global o un índice secundario.

Consola de administración de AWS

Para crear una tabla de DynamoDB y ajustar los valores de rendimiento en caliente a través de la consola:

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. Seleccione Crear tabla.
3. Elija un nombre de tabla, una clave de partición y una clave de clasificación (opcional).
4. En Configuración de la tabla, elija Personalizar configuración.

5. En el campo Rendimiento en caliente, seleccione Aumentar el rendimiento en caliente.
6. Ajuste las unidades de lectura por segundo y las unidades de escritura por segundo. Estas dos configuraciones definen el rendimiento máximo que la tabla puede gestionar al instante.
7. Continúe agregando los detalles de tabla restantes y, a continuación, seleccione Crear tabla.

AWS CLI

En el siguiente ejemplo de la AWS CLI se muestra cómo crear una tabla de DynamoDB con valores de rendimiento en caliente personalizados.

1. Ejecute la operación `create-table` para crear la siguiente tabla de DynamoDB.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S \
                            AttributeName=GameTitle,AttributeType=S \
                            AttributeName=TopScore,AttributeType=N  \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH
\"},
                                {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                \"Projection\":{
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 50,
                    \"WriteCapacityUnits\": 25
                },\"WarmThroughput\": {
                    \"ReadUnitsPerSecond\": 1987,
                    \"WriteUnitsPerSecond\": 543
                }
            }
        ]" \
    --warm-throughput ReadUnitsPerSecond=12345,WriteUnitsPerSecond=4567 \
```

```
    --region us-east-1
```

2. Verá una respuesta similar a la siguiente. La configuración de `WarmThroughput` se mostrará como `ReadUnitsPerSecond` y `WriteUnitsPerSecond`. El `Status` será `UPDATING` cuando se actualice el valor de rendimiento en caliente y `ACTIVE` cuando se establezca el nuevo valor de rendimiento en caliente.

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": 1730241788.779,
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 20,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
```

```
        "TableArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/GameScores",
        "TableId": "XXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "UserId"
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 50,
                    "WriteCapacityUnits": 25
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-east-1:XXXXXXXXXXXX:table/
GameScores/index/GameTitleIndex",
                "WarmThroughput": {
                    "ReadUnitsPerSecond": 1987,
                    "WriteUnitsPerSecond": 543,
                    "Status": "UPDATING"
                }
            }
        ],
        "DeletionProtectionEnabled": false,
        "WarmThroughput": {
            "ReadUnitsPerSecond": 12345,
            "WriteUnitsPerSecond": 4567,
            "Status": "UPDATING"
        }
```

```
    }
}
```

AWS SDK

En los siguientes ejemplos del SDK se muestra cómo crear una tabla de DynamoDB con valores de rendimiento en caliente personalizados.

Java

```
import software.amazon.awscdk.services.dynamodb.ProjectionType;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.Projection;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndex;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;
...

public static WarmThroughput buildWarmThroughput(final Long readUnitsPerSecond,
                                                 final Long writeUnitsPerSecond) {
    return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
            .build();
}
private static AttributeDefinition buildAttributeDefinition(final String
 attributeName,
                                                             final
 ScalarAttributeType scalarAttributeType) {
    return AttributeDefinition.builder()
            .attributeName(attributeName)
            .attributeType(scalarAttributeType)
            .build();
}
private static KeySchemaElement buildKeySchemaElement(final String attributeName,
                                                      final KeyType keyType) {
```

```
    return KeySchemaElement.builder()
            .attributeName(attributeName)
            .keyType(keyType)
            .build();
}
public static void createDynamoDBTable(DynamoDbClient ddb,
                                        String tableName,
                                        String partitionKey,
                                        String sortKey,
                                        String miscellaneousKeyAttribute,
                                        String nonKeyAttribute,
                                        Long tableReadCapacityUnits,
                                        Long tableWriteCapacityUnits,
                                        Long tableWarmReadUnitsPerSecond,
                                        Long tableWarmWriteUnitsPerSecond,
                                        String globalSecondaryIndexName,
                                        Long globalSecondaryIndexReadCapacityUnits,
                                        Long globalSecondaryIndexWriteCapacityUnits,
                                        Long
 globalSecondaryIndexWarmReadUnitsPerSecond,
                                        Long
 globalSecondaryIndexWarmWriteUnitsPerSecond) {

    // Define the table attributes
    final AttributeDefinition partitionKeyAttribute =
 buildAttributeDefinition(partitionKey, ScalarAttributeType.S);
    final AttributeDefinition sortKeyAttribute = buildAttributeDefinition(sortKey,
 ScalarAttributeType.S);
    final AttributeDefinition miscellaneousKeyAttributeDefinition =
 buildAttributeDefinition(miscellaneousKeyAttribute, ScalarAttributeType.N);
    final AttributeDefinition[] attributeDefinitions = {partitionKeyAttribute,
 sortKeyAttribute, miscellaneousKeyAttributeDefinition};

    // Define the table key schema
    final KeySchemaElement partitionKeyElement = buildKeySchemaElement(partitionKey,
 KeyType.HASH);
    final KeySchemaElement sortKeyElement = buildKeySchemaElement(sortKey,
 KeyType.RANGE);
    final KeySchemaElement[] keySchema = {partitionKeyElement, sortKeyElement};

    // Define the provisioned throughput for the table
    final ProvisionedThroughput provisionedThroughput =
 ProvisionedThroughput.builder()
            .readCapacityUnits(tableReadCapacityUnits)
```

```
            .writeCapacityUnits(tableWriteCapacityUnits)
            .build();

   // Define the Global Secondary Index (GSI)
   final KeySchemaElement globalSecondaryIndexPartitionKeyElement =
buildKeySchemaElement(sortKey, KeyType.HASH);
   final KeySchemaElement globalSecondaryIndexSortKeyElement =
buildKeySchemaElement(miscellaneousKeyAttribute, KeyType.RANGE);
   final KeySchemaElement[] gsiKeySchema =
{globalSecondaryIndexPartitionKeyElement, globalSecondaryIndexSortKeyElement};

   final Projection gsiProjection = Projection.builder()
           .projectionType(String.valueOf(ProjectionType.INCLUDE))
           .nonKeyAttributes(nonKeyAttribute)
           .build();
   final ProvisionedThroughput gsiProvisionedThroughput =
ProvisionedThroughput.builder()
           .readCapacityUnits(globalSecondaryIndexReadCapacityUnits)
           .writeCapacityUnits(globalSecondaryIndexWriteCapacityUnits)
           .build();
   // Define the warm throughput for the Global Secondary Index (GSI)
   final WarmThroughput gsiWarmThroughput =
buildWarmThroughput(globalSecondaryIndexWarmReadUnitsPerSecond,
globalSecondaryIndexWarmWriteUnitsPerSecond);
   final GlobalSecondaryIndex globalSecondaryIndex = GlobalSecondaryIndex.builder()
           .indexName(globalSecondaryIndexName)
           .keySchema(gsiKeySchema)
           .projection(gsiProjection)
           .provisionedThroughput(gsiProvisionedThroughput)
           .warmThroughput(gsiWarmThroughput)
           .build();

   // Define the warm throughput for the table
   final WarmThroughput tableWarmThroughput =
buildWarmThroughput(tableWarmReadUnitsPerSecond, tableWarmWriteUnitsPerSecond);

   final CreateTableRequest request = CreateTableRequest.builder()
           .tableName(tableName)
           .attributeDefinitions(attributeDefinitions)
           .keySchema(keySchema)
           .provisionedThroughput(provisionedThroughput)
           .globalSecondaryIndexes(globalSecondaryIndex)
           .warmThroughput(tableWarmThroughput)
           .build();
```

```
    CreateTableResponse response = ddb.createTable(request);
    System.out.println(response);
}
```

Python

```
from boto3 import resource
from botocore.exceptions import ClientError

def create_dynamodb_table_warm_throughput(table_name, partition_key,
 sort_key, misc_key_attr, non_key_attr, table_provisioned_read_units,
 table_provisioned_write_units, table_warm_reads, table_warm_writes, gsi_name,
 gsi_provisioned_read_units, gsi_provisioned_write_units, gsi_warm_reads,
 gsi_warm_writes, region_name="us-east-1"):
    """
    Creates a DynamoDB table with a warm throughput setting configured.

    :param table_name: The name of the table to be created.
    :param partition_key: The partition key for the table being created.
    :param sort_key: The sort key for the table being created.
    :param misc_key_attr: A miscellaneous key attribute for the table being created.
    :param non_key_attr: A non-key attribute for the table being created.
    :param table_provisioned_read_units: The newly created table's provisioned read
 capacity units.
    :param table_provisioned_write_units: The newly created table's provisioned
 write capacity units.
    :param table_warm_reads: The read units per second setting for the table's warm
 throughput.
    :param table_warm_writes: The write units per second setting for the table's
 warm throughput.
    :param gsi_name: The name of the Global Secondary Index (GSI) to be created on
 the table.
    :param gsi_provisioned_read_units: The configured Global Secondary Index (GSI)
 provisioned read capacity units.
    :param gsi_provisioned_write_units: The configured Global Secondary Index (GSI)
 provisioned write capacity units.
    :param gsi_warm_reads: The read units per second setting for the Global
 Secondary Index (GSI)'s warm throughput.
    :param gsi_warm_writes: The write units per second setting for the Global
 Secondary Index (GSI)'s warm throughput.
    :param region_name: The AWS Region name to target. defaults to us-east-1
    """
```

```
    try:
        ddb = resource('dynamodb', region_name)

        # Define the table attributes
        attribute_definitions = [
            { "AttributeName": partition_key, "AttributeType": "S" },
            { "AttributeName": sort_key, "AttributeType": "S" },
            { "AttributeName": misc_key_attr, "AttributeType": "N" }
        ]

        # Define the table key schema
        key_schema = [
            { "AttributeName": partition_key, "KeyType": "HASH" },
            { "AttributeName": sort_key, "KeyType": "RANGE" }
        ]

        # Define the provisioned throughput for the table
        provisioned_throughput = {
            "ReadCapacityUnits": table_provisioned_read_units,
            "WriteCapacityUnits": table_provisioned_write_units
        }

        # Define the global secondary index
        gsi_key_schema = [
            { "AttributeName": sort_key, "KeyType": "HASH" },
            { "AttributeName": misc_key_attr, "KeyType": "RANGE" }
        ]
        gsi_projection = {
            "ProjectionType": "INCLUDE",
            "NonKeyAttributes": [non_key_attr]
        }
        gsi_provisioned_throughput = {
            "ReadCapacityUnits": gsi_provisioned_read_units,
            "WriteCapacityUnits": gsi_provisioned_write_units
        }
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_warm_reads,
            "WriteUnitsPerSecond": gsi_warm_writes
        }
        global_secondary_indexes = [
            {
                "IndexName": gsi_name,
                "KeySchema": gsi_key_schema,
                "Projection": gsi_projection,
```

```
                "ProvisionedThroughput": gsi_provisioned_throughput,
                "WarmThroughput": gsi_warm_throughput
            }
        ]

        # Define the warm throughput for the table
        warm_throughput = {
            "ReadUnitsPerSecond": table_warm_reads,
            "WriteUnitsPerSecond": table_warm_writes
        }

        # Create the DynamoDB client and create the table
        response = ddb.create_table(
            TableName=table_name,
            AttributeDefinitions=attribute_definitions,
            KeySchema=key_schema,
            ProvisionedThroughput=provisioned_throughput,
            GlobalSecondaryIndexes=global_secondary_indexes,
            WarmThroughput=warm_throughput
        )

        print(response)
    except ClientError as e:
        print(f"Error creating table: {e}")
        raise e
```

Javascript

```
import { DynamoDBClient, CreateTableCommand } from "@aws-sdk/client-dynamodb";

async function createDynamoDBTableWithWarmThroughput(
  tableName,
  partitionKey,
  sortKey,
  miscKeyAttr,
  nonKeyAttr,
  tableProvisionedReadUnits,
  tableProvisionedWriteUnits,
  tableWarmReads,
  tableWarmWrites,
  indexName,
  indexProvisionedReadUnits,
  indexProvisionedWriteUnits,
```

```
  indexWarmReads,
  indexWarmWrites,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });
    const command = new CreateTableCommand({
      TableName: tableName,
      AttributeDefinitions: [
          { AttributeName: partitionKey, AttributeType: "S" },
          { AttributeName: sortKey, AttributeType: "S" },
          { AttributeName: miscKeyAttr, AttributeType: "N" },
      ],
      KeySchema: [
          { AttributeName: partitionKey, KeyType: "HASH" },
          { AttributeName: sortKey, KeyType: "RANGE" },
      ],
      ProvisionedThroughput: {
          ReadCapacityUnits: tableProvisionedReadUnits,
          WriteCapacityUnits: tableProvisionedWriteUnits,
      },
      WarmThroughput: {
          ReadUnitsPerSecond: tableWarmReads,
          WriteUnitsPerSecond: tableWarmWrites,
      },
      GlobalSecondaryIndexes: [
          {
            IndexName: indexName,
            KeySchema: [
                { AttributeName: sortKey, KeyType: "HASH" },
                { AttributeName: miscKeyAttr, KeyType: "RANGE" },
            ],
            Projection: {
                ProjectionType: "INCLUDE",
                NonKeyAttributes: [nonKeyAttr],
            },
            ProvisionedThroughput: {
                ReadCapacityUnits: indexProvisionedReadUnits,
                WriteCapacityUnits: indexProvisionedWriteUnits,
            },
            WarmThroughput: {
                ReadUnitsPerSecond: indexWarmReads,
                WriteUnitsPerSecond: indexWarmWrites,
            },
```

```
          },
        ],
      });
      const response = await ddbClient.send(command);
      console.log(response);
    } catch (error) {
      console.error(`Error creating table: ${error}`);
      throw error;
    }
}
```

Descripción del rendimiento en caliente de DynamoDB en diferentes escenarios

A continuación se presentan algunos escenarios diferentes que puede encontrar al trabajar con rendimiento en caliente de DynamoDB.

Temas

- Rendimiento en caliente y patrones de acceso desiguales
- Rendimiento en caliente para una tabla aprovisionada
- Rendimiento en caliente de una tabla bajo demanda
- Rendimiento en caliente para una tabla bajo demanda con el máximo rendimiento configurado

Rendimiento en caliente y patrones de acceso desiguales

Una tabla puede tener un rendimiento en caliente de 30 000 unidades de lectura por segundo y de 10 000 unidades de escritura por segundo, pero aún así podría experimentar una limitación en las lecturas o escrituras antes de llegar a esos valores. Se debe probablemente a una partición activa. Aunque DynamoDB puede seguir escalando para soportar un rendimiento prácticamente ilimitado, cada partición individual está limitada a 1000 unidades de escritura por segundo y 3000 unidades de lectura por segundo. Si la aplicación dirige demasiado tráfico a una pequeña parte de las particiones de la tabla, puede producirse una limitación incluso antes de alcanzar los valores de rendimiento en caliente de la tabla. Recomendamos seguir las prácticas recomendadas de DynamoDB para garantizar una escalabilidad fluida y evitar las particiones en caliente.

Rendimiento en caliente para una tabla aprovisionada

Considere una tabla aprovisionada que tiene un rendimiento en caliente de 30 000 unidades de lectura por segundo y 10 000 unidades de escritura por segundo, pero que actualmente tiene un rendimiento aprovisionado de 4000 RCU y 8000 WCU. Puede escalar instantáneamente el rendimiento aprovisionado de la tabla hasta 30 000 RCU o 10 000 WCU actualizando la configuración de rendimiento aprovisionada. A medida que aumente el rendimiento en caliente aprovisionado por encima de estos valores, el rendimiento en caliente se ajustará automáticamente a los nuevos valores superiores, porque habrá establecido un nuevo rendimiento pico. Por ejemplo, si establece el rendimiento en caliente aprovisionado en 50 000 RCU, el rendimiento en caliente aumentará a 50 000 unidades de lectura por segundo.

```
"ProvisionedThroughput":
    {
        "ReadCapacityUnits": 4000,
        "WriteCapacityUnits": 8000
    }
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 30000,
        "WriteUnitsPerSecond": 10000
    }
```

Rendimiento en caliente de una tabla bajo demanda

Una nueva tabla bajo demanda comienza con un rendimiento en caliente de 12 000 unidades de lectura por segundo y 4000 unidades de escritura por segundo. La tabla puede acomodar de forma instantánea un tráfico sostenido hasta estos niveles. Cuando las solicitudes superen las 12 000 unidades de lectura por segundo o las 4000 unidades de escritura por segundo, el rendimiento en caliente se ajustará automáticamente a valores superiores.

```
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 12000,
        "WriteUnitsPerSecond": 4000
    }
```

Rendimiento en caliente para una tabla bajo demanda con el máximo rendimiento configurado

Considere una tabla bajo demanda con un rendimiento en caliente de 30 000 unidades de lectura por segundo, pero con un rendimiento máximo configurado en 5000 unidades de solicitud de lectura (RRU). En este caso, el rendimiento de la tabla se limitará al máximo de las 5000 RRU que ha establecido. Cualquier solicitud de rendimiento que supere este máximo se limitará. No obstante, puede modificar en cualquier momento el rendimiento máximo específico de la tabla en función de las necesidades de la aplicación.

```
"OnDemandThroughput":
    {
        "MaxReadRequestUnits": 5000,
        "MaxWriteRequestUnits": 4000
    }
"WarmThroughput":
    {
        "ReadUnitsPerSecond": 30000,
        "WriteUnitsPerSecond": 10000
    }
```

Capacidad de ampliación y de adaptación de DynamoDB

Para minimizar las limitaciones debidas a las excepciones de rendimiento, DynamoDB utiliza la capacidad de ampliación para gestionar los picos de uso. DynamoDB utiliza la capacidad de adaptación para adaptarse a patrones de acceso irregulares.

Capacidad de ampliación

DynamoDB proporciona cierta flexibilidad para su aprovisionamiento de rendimiento con capacidad de ampliación. Cuando no se utiliza todo el rendimiento disponible, DynamoDB reserva una parte de esa capacidad no utilizada para realizar posteriormente ampliaciones de rendimiento durante los picos de uso. Con la capacidad de ráfaga, pueden realizarse correctamente solicitudes de lectura o escritura inesperadas que, de otro modo, habrían sido objeto de una limitación controlada.

En este momento, DynamoDB retiene hasta cinco minutos (300 segundos) de capacidad de lectura y escritura sin usar. Durante una ampliación de actividad de lectura o escritura ocasional, estas unidades de capacidad adicionales pueden consumirse muy rápidamente, incluso a más velocidad que la capacidad de rendimiento aprovisionada por segundo que se ha definido para la tabla.

DynamoDB también puede consumir capacidad de ampliación para el mantenimiento en segundo plano y otras tareas sin previo aviso.

Tenga en cuenta que estos detalles sobre la capacidad de ampliación podrían cambiar en el futuro.

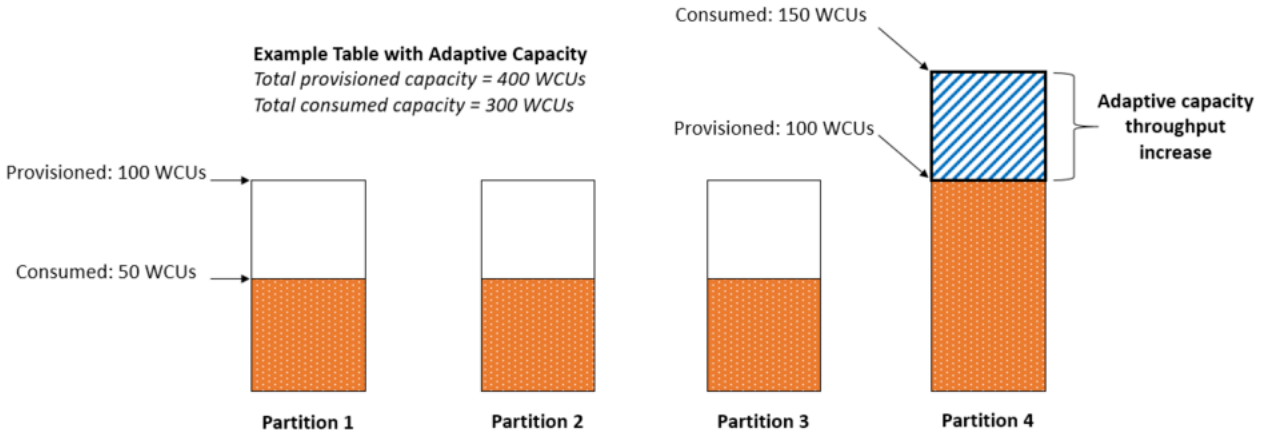
Capacidad de adaptación

DynamoDB distribuye automáticamente sus datos en particiones, que se almacenan en varios servidores en la Nube de AWS. No siempre es posible distribuir uniformemente la actividad de lectura y escritura. Si el acceso a los datos está desequilibrado, una partición "caliente" podría recibir un volumen mayor de tráfico de lectura y escritura que otras particiones. Dado que las operaciones de lectura y escritura en una partición se administran de forma independiente, se aplicará una limitación si una sola partición recibe más de 3000 operaciones de lectura o más de 1000 operaciones de escritura. De forma automática, la capacidad de adaptación hace que la capacidad de rendimiento aumente en las particiones que reciben más tráfico.

Para acomodar mejor los patrones de acceso desiguales, la capacidad de adaptación de DynamoDB permite que una aplicación pueda seguir leyendo particiones calientes y escribiendo en ellas sin que se produzca una limitación controlada, siempre que el tráfico no supere la capacidad total aprovisionada para la tabla o la capacidad máxima de la partición. De forma automática e instantánea, la capacidad de adaptación hace que la capacidad de rendimiento aumente en las particiones que reciben más tráfico.

El siguiente diagrama ilustra el funcionamiento de la capacidad de adaptación. La tabla de ejemplo está aprovisionada con 400 WCU distribuidas de manera uniforme entre cuatro particiones, lo que permite que cada una de ellas admita un máximo de 100 WCU por segundo. Cada una de las particiones 1, 2 y 3 recibe tráfico de escritura de 50 WCU/seg. La partición 4 recibe 150 WCU/seg. Esta partición caliente puede aceptar tráfico de escritura mientras haya capacidad de ampliación disponible, pero eventualmente limitará el tráfico que supere las 100 WCU/s.

La capacidad de adaptación de DynamoDB responde aumentando la capacidad de la partición 4 para que pueda soportar la mayor carga de trabajo de 150 WCU/s sin que se aplique una limitación.



La capacidad de adaptación está habilitada de forma automática para todas las tablas de DynamoDB sin coste adicional. No es necesario habilitarla o deshabilitarla de forma explícita.

Aislamiento de elementos de acceso frecuente

Si su aplicación dirige un tráfico alto hacia uno o dos elementos de forma desproporcionada, la capacidad de adaptación volverá a equilibrar sus particiones de manera que los elementos con acceso frecuente no residan en la misma partición. Este aislamiento de los elementos de acceso frecuente reduce la probabilidad de que se solicite la limitación controlada debida a que la carga de trabajo supere la cuota de rendimiento de una única partición. También puede dividir una colección de elementos en segmentos por clave de clasificación, siempre y cuando la colección de elementos no sea objeto de un seguimiento por aumento o disminución monótona de la clave de clasificación.

Si la aplicación genera habitualmente gran intensidad de tráfico dirigido a un solo elemento, la capacidad de adaptación puede reequilibrar los datos, de tal forma que una partición solo contenga ese elemento al que se accede con frecuencia. En este caso, DynamoDB puede proporcionar rendimiento hasta el máximo de la partición de 3000 RCU y 1000 WCU a la clave principal de ese elemento único. La capacidad de adaptación no dividirá las colecciones de elementos en varias particiones de la tabla cuando haya un índice secundario local en la tabla.

Aspectos a tener en cuenta al cambiar los modos de capacidad en DynamoDB

Cuando cree una tabla de DynamoDB, debe seleccionar el modo de capacidad bajo demanda o aprovisionada.

Puede cambiar las tablas del modo de capacidad aprovisionada al modo bajo demanda hasta cuatro veces en un periodo continuo de 24 horas. Las tablas pueden cambiar del modo bajo demanda al modo de capacidad aprovisionada en cualquier momento.

Temas

- Cambio del modo de capacidad aprovisionada al modo de capacidad bajo demanda
- Cambio del modo de capacidad bajo demanda al modo de capacidad aprovisionada

Cambio del modo de capacidad aprovisionada al modo de capacidad bajo demanda

En el modo aprovisionado, se establece la capacidad de lectura y escritura en función de las necesidades esperadas de la aplicación. Al actualizar una tabla en modo aprovisionado al modo bajo demanda, no necesita especificar el rendimiento de lectura y escritura que espera de su aplicación. DynamoDB bajo demanda ofrece precios de pago por solicitud para las solicitudes de lectura y escritura. De este modo, únicamente paga por aquello que utiliza, por lo que es fácil equilibrar los costos y el rendimiento. Si lo desea, también puede configurar el rendimiento máximo de lectura o escritura (o ambos) para tablas individuales bajo demanda e índices secundarios globales para ayudar a mantener limitados los costos y el uso. Para obtener más información sobre cómo configurar el rendimiento máximo para una tabla o índice específicos, consulte Rendimiento máximo de DynamoDB para las tablas bajo demanda.

Cuando se cambia del modo de capacidad aprovisionada al modo de capacidad bajo demanda, DynamoDB efectúa varios cambios en la estructura y las particiones de la tabla. Este proceso puede tardar varios minutos. Durante el periodo de cambio, la tabla proporciona el rendimiento acorde con las unidades de capacidad de lectura y de escritura aprovisionadas previamente.

Rendimiento inicial del modo de capacidad bajo demanda

Si recientemente ha cambiado una tabla existente al modo de capacidad bajo demanda por primera vez, la tabla tendrá la configuración del pico máximo anterior, aunque no haya atendido ningún tráfico en este modo.

A continuación, se muestran ejemplos de posibles escenarios:

- Cualquier tabla aprovisionada configurada por debajo de 4000 WCU y 12 000 RCU, que nunca se haya aprovisionado anteriormente para más cantidad. Cuando cambie esta tabla a la versión

bajo demanda por primera vez, DynamoDB se asegurará de que se escale horizontalmente para soportar al instante al menos 4000 unidades de escritura por segundo y 12 000 unidades de lectura por segundo.

- Una tabla aprovisionada configurada como 8000 WCU y 24 000 RCU. Cuando esta tabla se cambie a la versión bajo demanda, seguirá siendo capaz de soportar al menos 8000 unidades de escritura por segundo y 24 000 unidades de lectura por segundo en cualquier momento.
- Una tabla aprovisionada configurada con 8000 WCU y 24 000 RCU, que consumió 6000 unidades de escritura por segundo y 18 000 unidades de lectura por segundo durante un periodo prolongado. Cuando esta tabla se cambie a la versión bajo demanda, seguirá siendo capaz de soportar al menos 8000 unidades de escritura por segundo y 24 000 unidades de lectura por segundo. El tráfico anterior puede permitir además que la tabla mantenga niveles de tráfico mucho más altos sin limitaciones.
- Una tabla que anteriormente se aprovisionaba con 10 000 WCU y 10 000 RCU, pero que actualmente se aprovisiona con 10 RCU y 10 WCU. Cuando esta tabla se cambie a la versión bajo demanda, será capaz de soportar al menos 10 000 unidades de escritura por segundo y 10 000 unidades de lectura por segundo.

Configuración de escalado automático

Al actualizar una tabla del modo aprovisionado al modo bajo demanda:

- Si utiliza la consola, se eliminarán todos los ajustes de escalado automático (si los hay).
- Si utiliza la AWS CLI o el SDK de AWS, se conservarán todos los ajustes de escalado automático. Estos ajustes se pueden aplicar al actualizar la tabla de nuevo al modo aprovisionado de facturación.

Modo de capacidad de edición masiva en la consola de DynamoDB

Puede editar varias tablas de forma masiva para cambiar del modo de capacidad aprovisionada al modo de capacidad bajo demanda mediante la consola de DynamoDB. Para editar de forma masiva el modo de capacidad:

1. En la consola de DynamoDB, vaya a la página Tablas.
2. Seleccione las casillas de verificación de las tablas que desee actualizar al modo de capacidad bajo demanda.
3. Elija Acciones y, a continuación, seleccione Actualizar al modo de capacidad bajo demanda.

Esta operación masiva le permite cambiar de manera eficiente varias tablas al modo de capacidad bajo demanda sin tener que actualizar cada tabla de forma individual.

Cambio del modo de capacidad bajo demanda al modo de capacidad aprovisionada

Al cambiar del modo de capacidad bajo demanda al modo de capacidad aprovisionada, la tabla proporciona el rendimiento acorde con el tráfico máximo alcanzado anteriormente mientras la tabla estaba en el modo de capacidad bajo demanda.

Administración de la capacidad

Tenga en cuenta lo siguiente al actualizar una tabla del modo bajo demanda al modo aprovisionado:

- Si usa la AWS CLI o el SDK de AWS, elija la configuración de capacidad aprovisionada correctos de la tabla y de los índices secundarios globales tras haber consultado en Amazon CloudWatch el consumo histórico (métricas `ConsumedWriteCapacityUnits` y `ConsumedReadCapacityUnits`) para determinar los nuevos ajustes de rendimiento.

  > **Note**
  >
  > Si va a cambiar una tabla global al modo aprovisionado, fíjese en el consumo máximo en todas las réplicas regionales de las tablas base y en los índices secundarios globales para determinar los nuevos ajustes de rendimiento.

- Si va a cambiar del modo bajo demanda al modo aprovisionado, asegúrese de establecer unas unidades aprovisionadas iniciales lo suficientemente altas como para gestionar la capacidad de su tabla o índice durante la transición.

Administración de Auto Scaling

Al actualizar una tabla del modo bajo demanda al modo aprovisionado:

- Si utiliza la consola, le recomendamos habilitar el escalado automático con los valores predeterminados siguientes:
  - Objetivo de utilización: 70 %
  - Capacidad aprovisionada mínima: 5 unidades
  - Capacidad aprovisionada máxima: el máximo de la región

- Si utiliza la AWS CLI o el SDK, se conservan los ajustes anteriores de escalado automático (si los hay).

Programación con DynamoDB y los SDK de AWS

En este capítulo se tratan los temas relacionados con el desarrollador. Si lo que desea es ejecutar ejemplos de código, consulte Cómo ejecutar los ejemplos de código de esta guía para desarrolladores.

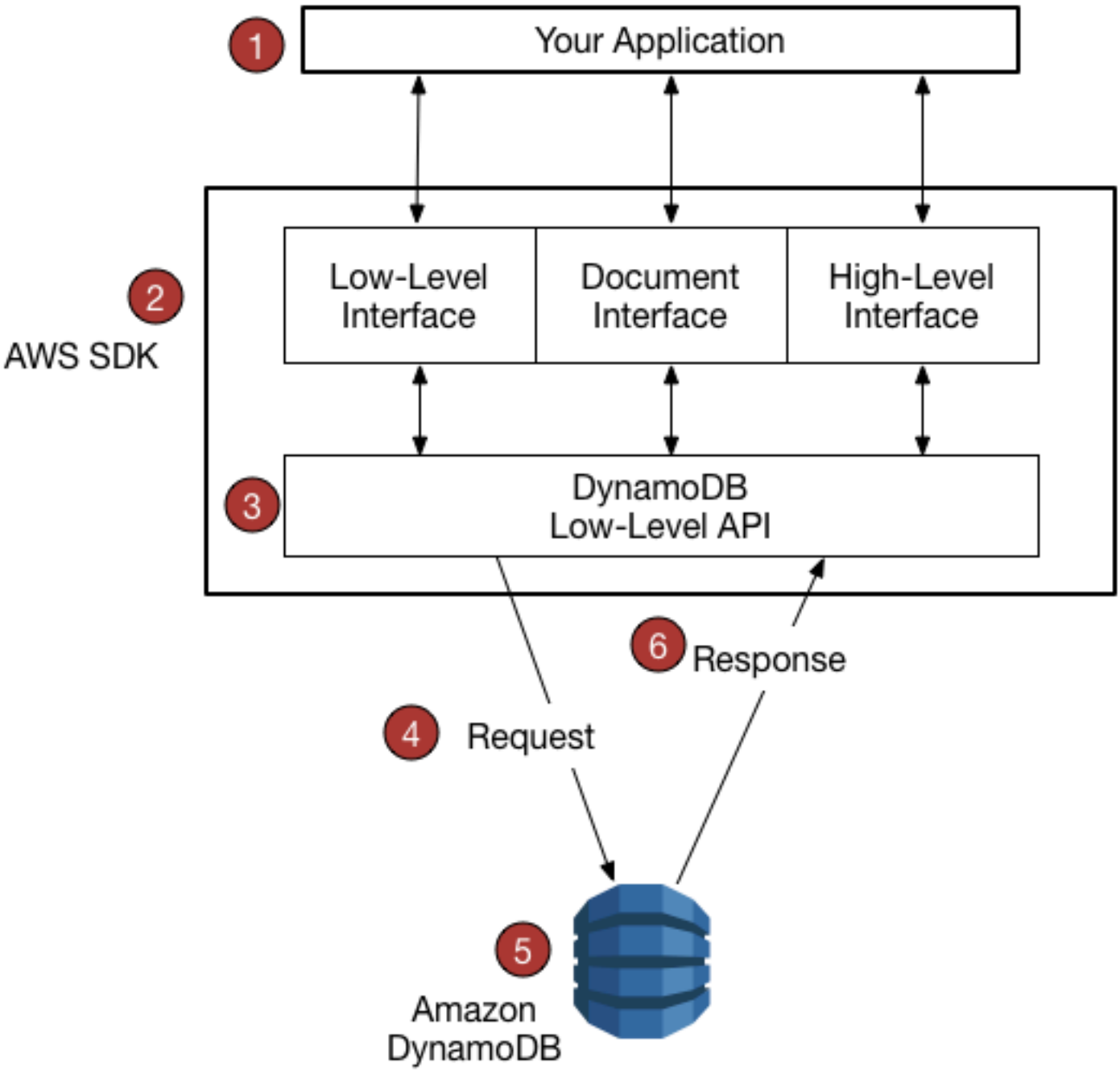
> **Note**
>
> En diciembre de 2017,AWS comenzó el proceso de migración de todos los puntos de enlace de Amazon DynamoDB para utilizar certificados seguros emitidos por Amazon Trust Services (ATS). Para obtener más información, consulte Solución de problemas de establecimiento de conexiones SSL/TLS con DynamoDB.

Temas

- Información general sobre la compatibilidad de los SDK de AWS con DynamoDB
- Programación de Amazon DynamoDB con Python y Boto3
- Programación de Amazon DynamoDB con JavaScript
- Programación de DynamoDB con AWS SDK for Java 2.x
- Control de errores con DynamoDB
- Uso de DynamoDB con un SDK de AWS

Información general sobre la compatibilidad de los SDK de AWS con DynamoDB

En el siguiente diagrama se ofrece información general sobre la programación de aplicaciones de Amazon DynamoDB con los SDK de AWS.



1. Puede escribir una aplicación utilizando un SDK de AWS para su lenguaje de programación.
2. Cada SDK de AWS proporciona una o varias interfaces de programación para trabajar con DynamoDB. Las interfaces específicas disponibles dependerán de qué lenguaje de programación y SDK de AWS se utilice. Las opciones son:
   - Interfaces de bajo nivel que funcionan con DynamoDB
   - Interfaces de documento que funcionan con DynamoDB
   - Interfaces de persistencia de objetos que funcionan con DynamoDB

- Interfaces de alto nivel

3. El SDK de AWS construye solicitudes HTTP(S) para usarlas con el API de bajo nivel de DynamoDB.
4. El SDK de AWS envía la solicitud al punto de enlace de DynamoDB.
5. DynamoDB ejecuta la solicitud. Si la solicitud se realiza correctamente, DynamoDB devuelve un código de respuesta HTTP 200 (OK). Si la solicitud no se puede realizar, DynamoDB devuelve un código de error HTTP y un mensaje de error.
6. El SDK de AWS procesa la respuesta y se la transmite a la aplicación.

Cada uno de los SDK de AWS presta servicios importantes a la aplicación, tales como los siguientes:

- Formatear las solicitudes HTTP(S) y serializar los parámetros de solicitud.
- Generar una firma criptográfica de cada solicitud.
- Reenviar la solicitud a un punto de enlace de DynamoDB y recibir las respuestas de DynamoDB.
- Extraer los resultados de esas respuestas.
- Implementar la lógica de reintento básica en caso de errores.

No es necesario escribir código para ninguna de estas tareas.

> **Note**
>
> Para obtener más información sobre los SDK de AWS, incluidas las instrucciones de instalación y la documentación, consulte las Herramientas para Amazon Web Services.

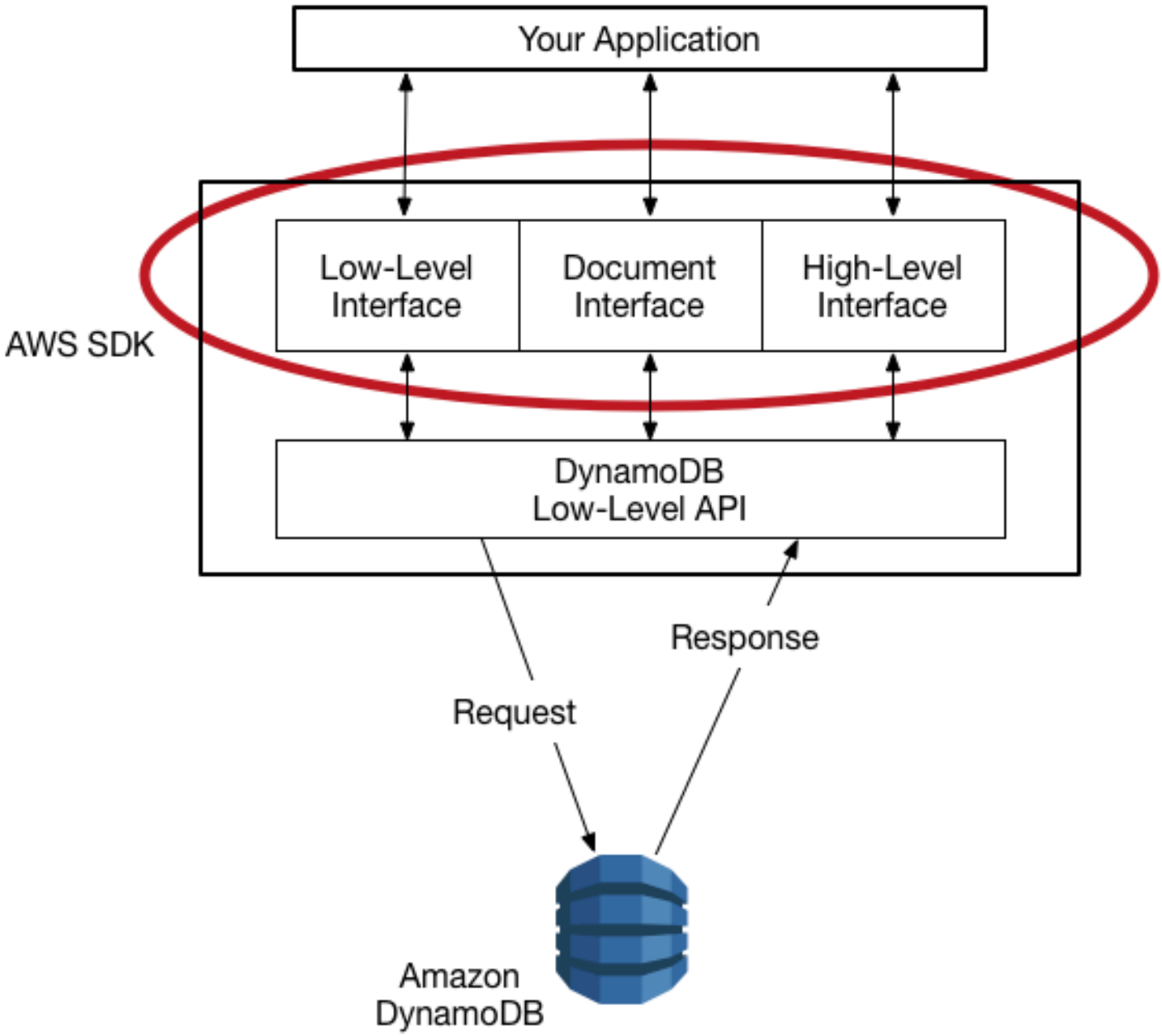
Compatibilidad del SDK con puntos de conexión basados en cuentas de AWS

AWS lanzará la compatibilidad con el SDK para puntos de conexión basados en cuentas de AWS para DynamoDB, empezando por el SDK de AWS para Java V1 el 4 de septiembre de 2024. Estos nuevos puntos de conexión ayudan a AWS a garantizar un alto rendimiento y escalabilidad. Los SDK actualizados utilizarán automáticamente los nuevos puntos de conexión, que tienen el formato `https://(account-id).ddb.(region).amazonaws.com`.

Si utiliza una sola instancia de un cliente del SDK para realizar solicitudes a varias cuentas, la aplicación tendrá menos oportunidades de reutilizar las conexiones. AWS recomienda modificar las aplicaciones para conectarse a un menor número de cuentas por instancia de cliente del SDK. Una alternativa es configurar el cliente del SDK para que siga utilizando los puntos de conexión regionales mediante la configuración `ACCOUNT_ID_ENDPOINT_MODE`, tal y como se documenta en la Guía de referencia de los SDK y las herramientas de AWS.

Interfaces programáticas que funcionan con DynamoDB

Cada SDK de AWS proporciona una o varias interfaces de programación para trabajar con Amazon DynamoDB. Estas interfaces abarcan desde sencillos encapsuladores de bajo nivel de DynamoDB hasta capas de persistencia orientadas a objetos. Las interfaces disponibles varían según el SDK de AWS y el lenguaje de programación que se utilice.



En la siguiente sección se resaltan algunas de las interfaces disponibles utilizando el AWS SDK para Java como ejemplo. (No todas las interfaces están disponibles en todos los SDK de AWS.)

Temas

- Interfaces de bajo nivel que funcionan con DynamoDB
- Interfaces de documento que funcionan con DynamoDB
- Interfaces de persistencia de objetos que funcionan con DynamoDB

Interfaces de bajo nivel que funcionan con DynamoDB

Cada SDK de AWS específico de un lenguaje ofrece una interfaz de bajo nivel para Amazon DynamoDB, con métodos que emulan del modo más parecido posible las solicitudes del API de bajo nivel de DynamoDB.

En algunos casos, es preciso identificar los tipos de datos de los atributos utilizando Descriptores de tipos de datos; por ejemplo, `S` si es una cadena o `N` si es un número.

> **Note**
>
> Hay interfaz de bajo nivel disponible en el SDK de AWS específico de cada idioma.

En el siguiente programa de Java se utiliza la interfaz de bajo nivel del AWS SDK para Java.

Ejemplo de interfaz de bajo nivel

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
    public static void main(String[] args) {
        final String usage = """
```

```
                Usage:
                    <tableName> <key> <keyVal>

                Where:
                    tableName - The Amazon DynamoDB table from which an item is
retrieved (for example, Music3).\s
                    key - The key used in the Amazon DynamoDB table (for example,
Artist).\s
                    keyval - The key value that represents the item to get (for
example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal, tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        getDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void getDynamoDBItem(DynamoDbClient ddb, String tableName, String
key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        GetItemRequest request = GetItemRequest.builder()
                .key(keyToGet)
                .tableName(tableName)
                .build();

        try {
            // If there is no matching item, GetItem does not return any data.
```

```
            Map<String, AttributeValue> returnedItem = ddb.getItem(request).item();
            if (returnedItem.isEmpty())
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Interfaces de documento que funcionan con DynamoDB

Muchos SDK de AWSproporcionan una interfaz de documentos, lo que permite realizar operaciones del plano de datos (crear, leer, actualizar, eliminar) en tablas e índices. Con una interfaz de documentos, no es preciso especificar Descriptores de tipos de datos. Los tipos de datos quedan implícitos en la propia semántica de los datos. Estos SDK de AWS también proporcionan métodos para convertir fácilmente documentos JSON a los tipos de datos nativos de Amazon DynamoDB y viceversa.

> **Note**
>
> Las interfaces de documento están disponibles en los SDK de AWS para Java, .NET, Node.js y JavaScript.

En el siguiente programa de Java se utiliza la interfaz de documentos del AWS SDK para Java. El programa crea un objeto `Table` que representa la tabla `Music` y, a continuación, solicita que el objeto utilice `GetItem` para recuperar una canción. Después, el programa imprime el año en que se lanzó la canción.

La clase `software.amazon.dynamodb.document.DynamoDB` implementa la interfaz de documentos de DynamoDB. Observe que `DynamoDB` actúa como encapsulador en torno al cliente de bajo nivel (`AmazonDynamoDB`).

Ejemplo de interfaz de documento

```
package com.amazonaws.codesamples.gsg;

import software.amazon.dynamodb.AmazonDynamoDB;
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import software.amazon.dynamodb.document.DynamoDB;
import software.amazon.dynamodb.document.GetItemOutcome;
import software.amazon.dynamodb.document.Table;

public class MusicDocumentDemo {

    public static void main(String[] args) {

        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
        DynamoDB docClient = new DynamoDB(client);

        Table table = docClient.getTable("Music");
        GetItemOutcome outcome = table.getItemOutcome(
                "Artist", "No One You Know",
                "SongTitle", "Call Me Today");

        int year = outcome.getItem().getInt("Year");
        System.out.println("The song was released in " + year);

    }
}
```

Interfaces de persistencia de objetos que funcionan con DynamoDB

Algunos SDK de AWS proporcionan una interfaz de persistencia de objetos en la que no se llevan a cabo directamente operaciones del plano de datos. En lugar de ello, se crean objetos que representan los elementos de las tablas y los índices de Amazon DynamoDB y solamente se interacciona con estos objetos. Esto le permite escribir código orientado a objetos, en lugar de código orientado a bases de datos.

> **Note**
>
> Las interfaces de persistencia de objetos están disponibles en los SDK de AWS para Java y .NET. Para obtener más información, consulte Interfaces de programación de nivel superior para DynamoDB de DynamoDB.

Ejemplo de interfaz de persistencia de objetos

```
import com.example.dynamodb.Customer;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import software.amazon.awssdk.enhanced.dynamodb.model.GetItemEnhancedRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
```

```
import com.example.dynamodb.Customer;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import software.amazon.awssdk.enhanced.dynamodb.model.GetItemEnhancedRequest;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;

/*
 * Before running this code example, create an Amazon DynamoDB table named Customer
 with these columns:
 *   - id - the id of the record that is the key. Be sure one of the id values is
 `id101`
 *   - custName - the customer name
 *   - email - the email value
 *   - registrationDate - an instant value when the item was added to the table. These
 values
 *                        need to be in the form of `YYYY-MM-DDTHH:mm:ssZ`, such as
 2022-07-11T00:00:00Z
 *
```

```
 * Also, ensure that you have set up your development environment, including your
 credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class EnhancedGetItem {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

        getItem(enhancedClient);
        ddb.close();
    }

    public static String getItem(DynamoDbEnhancedClient enhancedClient) {
        Customer result = null;
        try {
            DynamoDbTable<Customer> table = enhancedClient.table("Customer",
 TableSchema.fromBean(Customer.class));
            Key key = Key.builder()
                    .partitionValue("id101").sortValue("tred@noserver.com")
                    .build();

            // Get the item by using the key.
            result = table.getItem(
                    (GetItemEnhancedRequest.Builder requestBuilder) ->
 requestBuilder.key(key));
            System.out.println("******* The description value is " +
 result.getCustName());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return result.getCustName();
```

```
    }
}
```

Interfaces de programación de nivel superior para DynamoDB

Los SDK de AWS proporcionan aplicaciones con interfaces de bajo nivel para trabajar con Amazon DynamoDB. Estas clases y métodos del lado del cliente se corresponden directamente con los API de bajo nivel de DynamoDB. Sin embargo, muchos desarrolladores experimentan una sensación de desconexión, o discrepancia de impedancia cuando tienen que mapear tipos de datos complejos a los elementos de una tabla de base de datos. Con una interfaz de bajo nivel de base de datos, los desarrolladores deben escribir métodos para leer o escribir datos de objetos en las tablas de las bases de datos, y viceversa. La cantidad de código adicional necesaria para cada combinación de tipo de objeto y tabla de base de datos puede resultar abrumadora.

Para simplificar el desarrollo, los SDK de AWS para Java y .NET incluyen interfaces adicionales con niveles de abstracción superiores. Las interfaces de nivel superior para DynamoDB permiten definir las relaciones entre los objetos del programa del desarrollador y las tablas de base de datos en las que se almacenan esos datos de objetos. Una vez que se ha definido este mapeo, basta con llamar a métodos de objetos simples, como `save`, `load` o `delete`, y las operaciones de bajo nivel de DynamoDB subyacentes se invocan automáticamente si su intervención. Esto le permite escribir código orientado a objetos, en lugar de código orientado a bases de datos.

Las interfaces de programación de nivel superior para DynamoDB están disponibles en los SDK de AWS para Java y .NET.

Java

- Java 1.x: DynamoDBMapper
- Java 2.x: cliente mejorado de DynamoDB

.NET

- Uso del modelo de documento de .NET en DynamoDB
- Uso del modelo de persistencia de objetos de .NET y DynamoDB

Java 1.x: DynamoDBMapper

> **Note**
>
> El SDK para Java tiene dos versiones: 1.x y 2.x. El 12 de enero de 2024 se anunció el fin de soporte de la versión 1.x. Lo hará y está previsto que se dé por terminado el 31 de diciembre de 2025. Para nuevos desarrollos, recomendamos encarecidamente el uso de 2.x.

El AWS SDK para Java proporciona la clase `DynamoDBMapper` que permite mapear las clases del lado del cliente a las tablas de Amazon DynamoDB. Para usar `DynamoDBMapper`, se define la relación entre los elementos de una tabla de DynamoDB y sus instancias de objetos correspondientes en el código. La clase `DynamoDBMapper` permite realizar varias operaciones de creación, lectura, actualización y eliminación (CRUD, Create, Read, Update y Delete) en elementos y ejecutar consultas y análisis en tablas.

Temas

- Clase DynamoDBMapper
- Tipos de datos compatibles con asignador DynamoDBMapper para Java
- Anotaciones de Java para DynamoDB
- Ajustes de configuración opcionales para DynamoDBMapper
- DynamoDB y bloqueo positivo con el número de versión
- Asignación de datos arbitrarios en DynamoDB
- Ejemplos de DynamoDBMapper

> **Note**
>
> La clase `DynamoDBMapper` no permite crear, actualizar o eliminar tablas. Para realizar estas tareas, utilice la interfaz de bajo nivel del SDK para Java en su lugar.

El SDK para Java proporciona un conjunto de tipos de anotación para que pueda mapear las clases de tablas. Por ejemplo, tomemos una tabla `ProductCatalog` cuya clave de partición es `Id`.

```
ProductCatalog(Id, ...)
```

Puede mapear una clase de la aplicación cliente a la tabla `ProductCatalog` tal y como se muestra en el siguiente código Java. En este código se define un objeto Java estándar (POJO) denominado `CatalogItem` que utiliza anotaciones para mapear campos de objetos a nombres de atributos de DynamoDB.

Example

```
package com.amazonaws.codesamples;

import java.util.Set;

import software.amazon.dynamodb.datamodeling.DynamoDBAttribute;
import software.amazon.dynamodb.datamodeling.DynamoDBHashKey;
import software.amazon.dynamodb.datamodeling.DynamoDBIgnore;
import software.amazon.dynamodb.datamodeling.DynamoDBTable;

@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    private Integer id;
    private String title;
    private String ISBN;
    private Set<String> bookAuthors;
    private String someProp;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer id) {this.id = id; }

    @DynamoDBAttribute(attributeName="Title")
    public String getTitle() {return title; }
    public void setTitle(String title) { this.title = title; }

    @DynamoDBAttribute(attributeName="ISBN")
    public String getISBN() { return ISBN; }
    public void setISBN(String ISBN) { this.ISBN = ISBN; }

    @DynamoDBAttribute(attributeName="Authors")
    public Set<String> getBookAuthors() { return bookAuthors; }
    public void setBookAuthors(Set<String> bookAuthors) { this.bookAuthors =
 bookAuthors; }

    @DynamoDBIgnore
```

```
    public String getSomeProp() { return someProp; }
    public void setSomeProp(String someProp) { this.someProp = someProp; }
}
```

En el código anterior, la anotación `@DynamoDBTable` mapea la clase `CatalogItem` a la tabla `ProductCatalog`. Puede almacenar instancias de clases individuales como los elementos de la tabla. En la definición de clase, la anotación `@DynamoDBHashKey` mapea la propiedad `Id` a la clave principal.

De forma predeterminada, las propiedades de la clase se mapean a los atributos de la tabla que tienen el mismo nombre. Las propiedades `Title` e `ISBN` se mapean a los atributos de la tabla que tienen el mismo nombre.

La anotación `@DynamoDBAttribute` es opcional cuando el nombre del atributo de DynamoDB coincide con el nombre de la propiedad declarada en la clase. Si son distintos, use esta anotación con el parámetro `attributeName` para especificar qué atributo de DynamoDB se corresponde con esta propiedad.

En el ejemplo anterior, la anotación `@DynamoDBAttribute` se agrega a cada propiedad para garantizar que los nombres de las propiedades coincidan exactamente con las tablas creadas en un paso anterior y para mantener la coherencia con los nombres de atributo utilizados en otros ejemplos de código de esta guía.

La definición de clase puede tener propiedades que no se mapeen a ningún atributo de la tabla. Estas propiedades se identifican agregándoles la anotación `@DynamoDBIgnore`. En el ejemplo anterior, la propiedad `SomeProp` se ha marcado con la anotación `@DynamoDBIgnore`. Al cargar una instancia de `CatalogItem` en la tabla, la instancia de `DynamoDBMapper` no incluye la propiedad `SomeProp`. Tampoco el mapeador devuelve este atributo cuando se recupera un elemento de la tabla.

Después de haber definido la clase de mapeo, puede usar métodos `DynamoDBMapper` para escribir una instancia de esa clase en un elemento correspondiente de la tabla `Catalog`. En el siguiente ejemplo de código se muestra esta técnica.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

DynamoDBMapper mapper = new DynamoDBMapper(client);

CatalogItem item = new CatalogItem();
item.setId(102);
```

```
item.setTitle("Book 102 Title");
item.setISBN("222-2222222222");
item.setBookAuthors(new HashSet<String>(Arrays.asList("Author 1", "Author 2")));
item.setSomeProp("Test");

mapper.save(item);
```

En el siguiente ejemplo de código se muestra cómo recuperar el elemento y obtener acceso a algunos de sus atributos.

```
CatalogItem partitionKey = new CatalogItem();

partitionKey.setId(102);
DynamoDBQueryExpression<CatalogItem> queryExpression = new
 DynamoDBQueryExpression<CatalogItem>()
    .withHashKeyValues(partitionKey);

List<CatalogItem> itemList = mapper.query(CatalogItem.class, queryExpression);

for (int i = 0; i < itemList.size(); i++) {
    System.out.println(itemList.get(i).getTitle());
    System.out.println(itemList.get(i).getBookAuthors());
}
```

`DynamoDBMapper` ofrece un modo natural e intuitivo de usar los datos de DynamoDB en Java. También ofrece varias características integradas, tales como el bloqueo optimista, las transacciones ACID, la generación automática de valores de claves de partición y de orden y el control de versiones de objetos.

Clase DynamoDBMapper

La clase `DynamoDBMapper` es el punto de entrada de Amazon DynamoDB. Proporciona acceso a un punto de enlace de DynamoDB y le permite obtener acceso a sus datos en diversas tablas. También permite realizar varias operaciones de creación, lectura, actualización y eliminación (CRUD, Create, Read, Update y Delete) en elementos y ejecutar consultas y análisis en tablas. Esta clase proporciona los siguientes métodos para trabajar con DynamoDB.

Para obtener la documentación de Javadoc correspondiente, consulte DynamoDBMapper en la Referencia de la API de AWS SDK para Java.

Temas

- save
- carga
- eliminar
- consulta
- queryPage
- scan
- scanPage
- parallelScan
- batchSave
- batchLoad
- batchDelete
- batchWrite
- transactionWrite
- transactionLoad
- count
- generateCreateTableRequest
- createS3Link
- getS3ClientCache

save

Guarda el objeto especificado en la tabla. El objeto que se desea guardar es el único parámetro obligatorio para este método. Puede usar el objeto `DynamoDBMapperConfig` para proporcionar parámetros de configuración opcionales.

Si no hay un elemento que tenga la misma clave principal, este método crea un nuevo elemento en la tabla. Si hay un elemento que tiene la misma clave principal, lo actualiza. Si las claves de partición y orden son de tipo String y se han anotado con `@DynamoDBAutoGeneratedKey`, se les asigna un identificador universal único (UUID) aleatorio si se deja sin inicializar. Los campos de versión que se anotan con `@DynamoDBVersionAttribute` se incrementan en una unidad. Además, si se actualiza un campo de versión o se genera una clave, el objeto que se ha pasado se actualiza como consecuencia de la operación.

De forma predeterminada, solo se actualizan los atributos correspondientes a propiedades de clases mapeadas. Ningún atributo existente adicional de un elemento se ve afectado. Sin embargo, si especifica `SaveBehavior.CLOBBER`, puede forzar que se sobrescriba el elemento completo.

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER).build();

mapper.save(item, config);
```

Si el control de versiones está habilitado, las versiones del elemento del lado del cliente y del lado del servidor deben coincidir. Sin embargo, no es preciso que coincidan las versiones si se utiliza la opción `SaveBehavior.CLOBBER`. Para obtener más información sobre el control de versiones, consulte DynamoDB y bloqueo positivo con el número de versión.

carga

Recupera un elemento de una tabla. Es preciso proporcionar la clave principal del elemento que se desea recuperar. Puede usar el objeto `DynamoDBMapperConfig` para proporcionar parámetros de configuración opcionales. Por ejemplo, si lo desea puede solicitar lecturas de consistencia alta para asegurarse de que este método recupere solamente los valores más recientes de los elementos, como se muestra en la siguiente instrucción de Java.

```
DynamoDBMapperConfig config = DynamoDBMapperConfig.builder()
    .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT).build();

CatalogItem item = mapper.load(CatalogItem.class, item.getId(), config);
```

De forma predeterminada, DynamoDB devuelve el elemento cuyos valores presentan consistencia final. Para obtener más información sobre el modelo de consistencia final de DynamoDB, consulte Coherencia de lectura de DynamoDB.

eliminar

Elimina un elemento de la tabla. Debe pasar una instancia de objeto de la clase mapeada.

Si el control de versiones está habilitado, las versiones del elemento del lado del cliente y del lado del servidor deben coincidir. Sin embargo, no es preciso que coincidan las versiones si se utiliza la opción `SaveBehavior.CLOBBER`. Para obtener más información sobre el control de versiones, consulte DynamoDB y bloqueo positivo con el número de versión.

consulta

Consulta una tabla o un índice secundario.

Supongamos que tenemos una tabla `Reply` en la que se almacenan respuestas de conversaciones. Para cada tema de conversación puede haber cero o más respuestas. La clave principal de la tabla `Reply` consta de los campos `Id` y `ReplyDateTime`, donde `Id` es la clave de partición y `ReplyDateTime` es la clave de orden de la clave principal.

```
Reply ( Id, ReplyDateTime, ... )
```

Supongamos que creamos un mapeo entre una clase `Reply` y la tabla `Reply` correspondiente de DynamoDB. En el siguiente código Java se usa `DynamoDBMapper` para buscar todas las respuestas de las últimas dos semanas para un tema de conversación concreto.

Example

```
String forumName = "&DDB;";
String forumSubject = "&DDB; Thread 1";
String partitionKey = forumName + "#" + forumSubject;

long twoWeeksAgoMilli = (new Date()).getTime() - (14L*24L*60L*60L*1000L);
Date twoWeeksAgo = new Date();
twoWeeksAgo.setTime(twoWeeksAgoMilli);
SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'");
String twoWeeksAgoStr = df.format(twoWeeksAgo);

Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":v1", new AttributeValue().withS(partitionKey));
eav.put(":v2",new AttributeValue().withS(twoWeeksAgoStr.toString()));

DynamoDBQueryExpression<Reply> queryExpression = new DynamoDBQueryExpression<Reply>()
    .withKeyConditionExpression("Id = :v1 and ReplyDateTime > :v2")
    .withExpressionAttributeValues(eav);

List<Reply> latestReplies = mapper.query(Reply.class, queryExpression);
```

La consulta devuelve una colección de objetos `Reply`.

De forma predeterminada, el método `query` devuelve una colección de "carga diferida". Inicialmente devuelve una sola página de resultados y, a continuación, realiza una llamada de servicio para

obtener la página siguiente si es necesario. Para obtener todos los elementos coincidentes, recorra en iteración la colección `latestReplies`.

Tenga en cuenta que llamar al método `size()` en la colección cargará todos los resultados para proporcionar un recuento preciso. Esto puede provocar que se consuma una gran cantidad de rendimiento aprovisionado y en una tabla muy grande incluso podría agotar toda la memoria en su JVM.

Para consultar un índice, antes es preciso modelarlo como clase de mapeador. Supongamos que la tabla `Reply` tiene un índice secundario global denominado PostedBy-Message-Index. La clave de partición de este índice es `PostedBy` y la de orden, `Message`. La definición de clase de un elemento del índice tendría el siguiente aspecto:

```
@DynamoDBTable(tableName="Reply")
public class PostedByMessage {
    private String postedBy;
    private String message;

    @DynamoDBIndexHashKey(globalSecondaryIndexName = "PostedBy-Message-Index",
 attributeName = "PostedBy")
    public String getPostedBy() { return postedBy; }
    public void setPostedBy(String postedBy) { this.postedBy = postedBy; }

    @DynamoDBIndexRangeKey(globalSecondaryIndexName = "PostedBy-Message-Index",
 attributeName = "Message")
    public String getMessage() { return message; }
    public void setMessage(String message) { this.message = message; }

   // Additional properties go here.
}
```

La anotación `@DynamoDBTable` indica que este índice está asociado a la tabla `Reply`. La anotación `@DynamoDBIndexHashKey` se refiere a la clave de partición (PostedBy) del índice y la anotación `@DynamoDBIndexRangeKey`, a su clave de ordenación (Message).

Ahora, puede usar `DynamoDBMapper` para consultar el índice y recuperar un subconjunto de los mensajes publicados por un usuario determinado. No necesita especificar el nombre del índice si no tiene asignaciones conflictivas entre tablas e índices y las asignaciones ya están hechas en el mapeador. El mapeador deducirá en función de la clave principal y la clave de clasificación. El siguiente código consulta el índice secundario global. Es imprescindible especificar

`withConsistentRead(false)`, ya que los índices secundarios globales admiten las lecturas consistentes finales, pero no las de consistencia alta.

```
HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":v1",  new AttributeValue().withS("User A"));
eav.put(":v2",  new AttributeValue().withS("DynamoDB"));

DynamoDBQueryExpression<PostedByMessage> queryExpression = new
 DynamoDBQueryExpression<PostedByMessage>()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false)
    .withKeyConditionExpression("PostedBy = :v1 and begins_with(Message, :v2)")
    .withExpressionAttributeValues(eav);

List<PostedByMessage> iList =  mapper.query(PostedByMessage.class, queryExpression);
```

La consulta devuelve una colección de objetos `PostedByMessage`.

queryPage

Consulta una tabla o un índice secundario y devuelve una sola página de resultados coincidentes. Al igual que con el método `query`, es preciso especificar un valor de clave de partición y un filtro de consulta que se aplica al atributo de clave de ordenación. Sin embargo, `queryPage` solamente devuelve la primera "página" de datos; es decir, la cantidad de datos que se ajusta a 1 MB

scan

Examina una tabla o un índice secundario completos. Si lo desea, puede especificar una expresión `FilterExpression` para filtrar el conjunto de resultados.

Supongamos que tenemos una tabla `Reply` en la que se almacenan respuestas de conversaciones. Para cada tema de conversación puede haber cero o más respuestas. La clave principal de la tabla `Reply` consta de los campos `Id` y `ReplyDateTime`, donde `Id` es la clave de partición y `ReplyDateTime` es la clave de orden de la clave principal.

```
Reply ( Id, ReplyDateTime, ... )
```

Si ha mapeado una clase de Java a la tabla `Reply`, puede usar `DynamoDBMapper` para examinar la tabla. Por ejemplo, en el siguiente código Java se examina toda la tabla `Reply` y únicamente se devuelven las respuestas de un año determinado.

Example

```
HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":v1", new AttributeValue().withS("2015"));

DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withFilterExpression("begins_with(ReplyDateTime,:v1)")
    .withExpressionAttributeValues(eav);

List<Reply> replies =  mapper.scan(Reply.class, scanExpression);
```

De forma predeterminada, el método `scan` devuelve una colección de "carga diferida". Inicialmente devuelve una sola página de resultados y, a continuación, realiza una llamada de servicio para obtener la página siguiente si es necesario. Para obtener todos los elementos coincidentes, recorra en iteración la colección `replies`.

Tenga en cuenta que llamar al método `size()` en la colección cargará todos los resultados para proporcionar un recuento preciso. Esto puede provocar que se consuma una gran cantidad de rendimiento aprovisionado y en una tabla muy grande incluso podría agotar toda la memoria en su JVM.

Para examinar un índice, antes es preciso modelarlo como clase de mapeador. Supongamos que la tabla `Reply` tiene un índice secundario global denominado `PostedBy-Message-Index`. La clave de partición de este índice es `PostedBy` y la de orden, `Message`. En la sección consulta se muestra una clase de mapeador para este índice. Usa las anotaciones `@DynamoDBIndexHashKey` y `@DynamoDBIndexRangeKey` para especificar la clave de ordenación y la de partición del índice.

En el siguiente ejemplo de código se examina `PostedBy-Message-Index`. No se utiliza ningún filtro de examen, por lo que se devuelven todos los elementos del índice.

```
DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withIndexName("PostedBy-Message-Index")
    .withConsistentRead(false);

    List<PostedByMessage> iList =  mapper.scan(PostedByMessage.class, scanExpression);
    Iterator<PostedByMessage> indexItems = iList.iterator();
```

scanPage

Examina una tabla o un índice secundario y devuelve una sola página de resultados coincidentes. Al igual que sucede con el método `scan`, si lo desea, puede especificar una expresión

`FilterExpression` para filtrar el conjunto de resultados. Sin embargo, `scanPage` solamente devuelve la primera "página" de datos, es decir, la cantidad de datos que caben en 1 MB.

parallelScan

Realiza un examen en paralelo de una tabla o un índice secundario completos. Se especifica un número de segmentos lógicos de la tabla, junto con una expresión de examen para filtrar los resultados. `parallelScan` divide la tarea de examen entre varios procesos de trabajo, uno para cada segmento lógico. Los procesos de trabajo examinan los datos en paralelo y devuelven los resultados.

En el siguiente ejemplo de código Java se realiza un examen paralelo de la tabla `Product`.

```
int numberOfThreads = 4;

Map<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
eav.put(":n", new AttributeValue().withN("100"));

DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
    .withFilterExpression("Price <= :n")
    .withExpressionAttributeValues(eav);

List<Product> scanResult = mapper.parallelScan(Product.class, scanExpression,
 numberOfThreads);
```

batchSave

Guarda objetos en una o varias tablas mediante una o varias llamadas al método `AmazonDynamoDB.batchWriteItem`. Este método no proporciona garantías de transacción.

En el siguiente código Java se guardan dos elementos (libros) en la tabla `ProductCatalog`.

```
Book book1 = new Book();
book1.setId(901);
book1.setProductCategory("Book");
book1.setTitle("Book 901 Title");

Book book2 = new Book();
book2.setId(902);
book2.setProductCategory("Book");
book2.setTitle("Book 902 Title");
```

```
mapper.batchSave(Arrays.asList(book1, book2));
```

batchLoad

Recupera varios elementos de una o varias tablas mediante sus claves principales.

En el siguiente código Java se recuperan dos elementos de dos tablas distintas.

```
ArrayList<Object> itemsToGet = new ArrayList<Object>();

ForumItem forumItem = new ForumItem();
forumItem.setForumName("Amazon DynamoDB");
itemsToGet.add(forumItem);

ThreadItem threadItem = new ThreadItem();
threadItem.setForumName("Amazon DynamoDB");
threadItem.setSubject("Amazon DynamoDB thread 1 message text");
itemsToGet.add(threadItem);

Map<String, List<Object>> items = mapper.batchLoad(itemsToGet);
```

batchDelete

Elimina objetos de una o varias tablas mediante una o varias llamadas al método `AmazonDynamoDB.batchWriteItem`. Este método no proporciona garantías de transacción.

En el siguiente código Java se eliminan dos elementos (libros) de la tabla `ProductCatalog`.

```
Book book1 = mapper.load(Book.class, 901);
Book book2 = mapper.load(Book.class, 902);
mapper.batchDelete(Arrays.asList(book1, book2));
```

batchWrite

Guarda o elimina objetos en una o varias tablas mediante una o varias llamadas al método `AmazonDynamoDB.batchWriteItem`. Este método no proporciona garantías de transacción ni admite el control de versiones (colocaciones o eliminaciones condicionales).

En el siguiente código Java se escribe un nuevo elemento en la tabla `Forum`, se escribe un nuevo elemento en la tabla `Thread` y se elimina un elemento de la tabla `ProductCatalog`.

```
// Create a Forum item to save
```

```
Forum forumItem = new Forum();
forumItem.setName("Test BatchWrite Forum");

// Create a Thread item to save
Thread threadItem = new Thread();
threadItem.setForumName("AmazonDynamoDB");
threadItem.setSubject("My sample question");

// Load a ProductCatalog item to delete
Book book3 = mapper.load(Book.class, 903);

List<Object> objectsToWrite = Arrays.asList(forumItem, threadItem);
List<Book> objectsToDelete = Arrays.asList(book3);

mapper.batchWrite(objectsToWrite, objectsToDelete);
```

transactionWrite

Guarda o elimina objetos en una o varias tablas mediante una llamada al método `AmazonDynamoDB.transactWriteItems`.

Para ver una lista de excepciones específicas de la transacción, consulte Errores de TransactWriteItems.

Para obtener más información sobre las transacciones de DynamoDB y las garantías proporcionadas de atomicidad, coherencia, aislamiento y durabilidad (ACID), consulte Amazon DynamoDB Transactions.

> **Note**
>
> Este método no admite lo siguiente:
>
> - DynamoDBMapperConfig.SaveBehavior.

El siguiente código Java escribe un nuevo elemento en cada una de las tablas `Forum` y `Thread` de un modo transaccional.

```
Thread s3ForumThread = new Thread();
s3ForumThread.setForumName("S3 Forum");
s3ForumThread.setSubject("Sample Subject 1");
```

```
s3ForumThread.setMessage("Sample Question 1");

Forum s3Forum = new Forum();
s3Forum.setName("S3 Forum");
s3Forum.setCategory("Amazon Web Services");
s3Forum.setThreads(1);

TransactionWriteRequest transactionWriteRequest = new TransactionWriteRequest();
transactionWriteRequest.addPut(s3Forum);
transactionWriteRequest.addPut(s3ForumThread);
mapper.transactionWrite(transactionWriteRequest);
```

transactionLoad

Carga objetos de una o varias tablas mediante una llamada al método `AmazonDynamoDB.transactGetItems`.

Para ver una lista de excepciones específicas de transacciones, consulte Errores de TransactGetItems.

Para obtener más información sobre las transacciones de DynamoDB y las garantías proporcionadas de atomicidad, coherencia, aislamiento y durabilidad (ACID), consulte Amazon DynamoDB Transactions.

El siguiente código Java carga un elemento en cada una de las tablas `Forum` y `Thread` de un modo transaccional.

```
Forum dynamodbForum = new Forum();
dynamodbForum.setName("DynamoDB Forum");
Thread dynamodbForumThread = new Thread();
dynamodbForumThread.setForumName("DynamoDB Forum");

TransactionLoadRequest transactionLoadRequest = new TransactionLoadRequest();
transactionLoadRequest.addLoad(dynamodbForum);
transactionLoadRequest.addLoad(dynamodbForumThread);
mapper.transactionLoad(transactionLoadRequest);
```

count

Evalúa la expresión de examen especificada y devuelve el recuento de elementos coincidentes. No se devuelven datos de elementos.

generateCreateTableRequest

Analiza una clase de objeto Java estándar (POJO) que representa una tabla de DynamoDB y devuelve una solicitud de `CreateTableRequest` para esa tabla.

createS3Link

Crea un enlace a un objeto en Amazon S3. Debe especificar un nombre de bucket y un nombre de clave para identificar el objeto en el bucket de forma exclusiva.

Para usar `createS3Link`, la clase de mapeador debe definir métodos getter y setter. Esto se ilustra en el siguiente ejemplo de código con la adición de un nuevo atributo y de métodos getter/setter a la clase `CatalogItem`.

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    ...

    public S3Link productImage;

    ....

    @DynamoDBAttribute(attributeName = "ProductImage")
    public S3Link getProductImage() {
            return productImage;
    }

    public void setProductImage(S3Link productImage) {
        this.productImage = productImage;
    }

...
}
```

En el siguiente código Java se define un nuevo elemento para escribirlo en la tabla `Product`. El elemento incluye un enlace a la imagen de un producto; los datos de la imagen se cargan en Amazon S3.

```
CatalogItem item = new CatalogItem();

item.setId(150);
```

```
item.setTitle("Book 150 Title");

String amzn-s3-demo-bucket = "amzn-s3-demo-bucket";
String myS3Key = "productImages/book_150_cover.jpg";
item.setProductImage(mapper.createS3Link(amzn-s3-demo-bucket, myS3Key));

item.getProductImage().uploadFrom(new File("/file/path/book_150_cover.jpg"));

mapper.save(item);
```

La clase `S3Link` proporciona muchos métodos más para manipular objetos en Amazon S3. Para obtener más información, consulte los javadocs en Class `S3Link`.

getS3ClientCache

Devuelve el objeto `S3ClientCache` subyacente para obtener acceso a Amazon S3. Un objeto `S3ClientCache` es un mapa inteligente para objetos `AmazonS3Client`. Si tiene varios clientes, `S3ClientCache` puede ayudarle a organizarlos por región de AWS y a crear nuevos clientes de Amazon S3 en diferido.

Tipos de datos compatibles con asignador DynamoDBMapper para Java

En esta sección se describen los tipos de datos arbitrarios, las colecciones y los tipos de datos de Java primitivos compatibles en Amazon DynamoDB.

Amazon DynamoDB admite los siguientes tipos de datos y clases contenedoras Java primitivos.

- `String`
- `Boolean`, `boolean`
- `Byte`, `byte`
- `Date` (como una cadena ISO_8601 con precisión de milisegundos, convertida a UTC)
- `Calendar` (como una cadena ISO_8601 con precisión de milisegundos, convertida a UTC)
- `Long`, `long`
- `Integer`, `int`
- `Double`, `double`
- `Float`, `float`
- `BigDecimal`

- `BigInteger`

> **Note**
>
> - Para obtener más información sobre las reglas de asignación de nombres de DynamoDB y los distintos tipos de datos admitidos, consulte Tipos de datos y reglas de nomenclatura admitidos en Amazon DynamoDB.
> - DynamoDBMapper admite valores binarios vacíos.
> - Los valores de cadena vacíos son compatibles con AWS SDK for Java 2.x.
>
> En AWS SDK para Java 1.x, DynamoDBMapper admite leer valores de atributo de cadena vacíos; sin embargo, no escribirá valores de atributo de cadena vacíos, ya que estos atributos se eliminan de la solicitud.

DynamoDB admite los tipos de colecciones de Java Set, List y Map. En la tabla siguiente se resume el mapeo de estos tipos de Java a los tipos de DynamoDB.

Tipo de Java	Tipo DynamoDB
Todos los tipos de números	`N` (tipo Number)
Cadenas	`S` (tipo String)
Booleano	`BOOL` (tipo booleano), 0 o 1.
ByteBuffer	`B` (tipo Binary)
Date	`S` (tipo String). Los valores Date se almacenan como cadenas con formato ISO-8601.
Tipos de colección Set	`SS` (tipo String Set), `NS` (tipo Number Set) o `BS` (tipo Binary Set)

La interfaz `DynamoDBTypeConverter` permite mapear sus propios tipos de datos arbitrarios a un tipo de datos que sea compatible de forma nativa con DynamoDB. Para obtener más información, consulte Asignación de datos arbitrarios en DynamoDB.

Anotaciones de Java para DynamoDB

En esta sección se describen las anotaciones que están disponibles para mapear las clases y las propiedades a las tablas y los atributos en Amazon DynamoDB.

Para obtener la documentación de Javadoc correspondiente, consulte Annotation Types Summary (Resumen de tipos de anotaciones) en la Referencia de la API AWS SDK para Java.

> **Note**
>
> En las anotaciones siguientes, solo son obligatorias `DynamoDBTable` y `DynamoDBHashKey`.

Temas

- DynamoDBAttribute
- DynamoDBAutoGeneratedKey
- DynamoDBAutoGeneratedTimestamp
- DynamoDBDocument
- DynamoDBHashKey
- DynamoDBIgnore
- DynamoDBIndexHashKey
- DynamoDBIndexRangeKey
- DynamoDBRangeKey
- DynamoDBTable
- DynamoDBTypeConverted
- DynamoDBTyped
- DynamoDBVersionAttribute

DynamoDBAttribute

Mapea una propiedad a un atributo de tabla. De forma predeterminada, cada propiedad de clase se mapea a un atributo de elemento con el mismo nombre. Sin embargo, si los nombres no son iguales, puede utilizar esta anotación para mapear una propiedad al atributo. En el siguiente fragmento de

Java, `DynamoDBAttribute` mapea la propiedad `BookAuthors` al nombre de atributo `Authors` de la tabla.

```
@DynamoDBAttribute(attributeName = "Authors")
public List<String> getBookAuthors() { return BookAuthors; }
public void setBookAuthors(List<String> BookAuthors) { this.BookAuthors =
 BookAuthors; }
```

`DynamoDBMapper` utiliza `Authors` como nombre de atributo al guardar el objeto en la tabla.

DynamoDBAutoGeneratedKey

Marca una propiedad de clave de partición o de clave de ordenación como generada automáticamente. `DynamoDBMapper` genera un UUID aleatorio al guardar estos atributos. Solo se pueden marcar propiedades de tipo String como claves generadas automáticamente.

El siguiente ejemplo muestra el uso de claves generadas automáticamente.

```
@DynamoDBTable(tableName="AutoGeneratedKeysExample")
public class AutoGeneratedKeys {
    private String id;
    private String payload;

    @DynamoDBHashKey(attributeName = "Id")
    @DynamoDBAutoGeneratedKey
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }

    @DynamoDBAttribute(attributeName="payload")
    public String getPayload() { return this.payload; }
    public void setPayload(String payload) { this.payload = payload; }

    public static void saveItem() {
        AutoGeneratedKeys obj = new AutoGeneratedKeys();
        obj.setPayload("abc123");

        // id field is null at this point
        DynamoDBMapper mapper = new DynamoDBMapper(dynamoDBClient);
        mapper.save(obj);

        System.out.println("Object was saved with id " + obj.getId());
    }
```

```
}
```

DynamoDBAutoGeneratedTimestamp

Genera automáticamente una marca de tiempo.

```
@DynamoDBAutoGeneratedTimestamp(strategy=DynamoDBAutoGenerateStrategy.ALWAYS)
public Date getLastUpdatedDate() { return lastUpdatedDate; }
public void setLastUpdatedDate(Date lastUpdatedDate) { this.lastUpdatedDate =
 lastUpdatedDate; }
```

La estrategia de generación automática también puede definirse si se proporciona un atributo de estrategia. El valor predeterminado es `ALWAYS`.

DynamoDBDocument

Indica que una clase se puede serializar como un documento de Amazon DynamoDB.

Por ejemplo, supongamos que desea mapear un documento JSON a un atributo de DynamoDB de tipo Map (`M`). En el siguiente ejemplo de código se define un elemento que contiene un atributo anidado (Pictures) de tipo Map.

```
public class ProductCatalogItem {

    private Integer id;  //partition key
    private Pictures pictures;
    /* ...other attributes omitted... */

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id;}
    public void setId(Integer id) {this.id = id;}

    @DynamoDBAttribute(attributeName="Pictures")
    public Pictures getPictures() { return pictures;}
    public void setPictures(Pictures pictures) {this.pictures = pictures;}

    // Additional properties go here.

    @DynamoDBDocument
    public static class Pictures {
        private String frontView;
        private String rearView;
```

```
        private String sideView;

        @DynamoDBAttribute(attributeName = "FrontView")
        public String getFrontView() { return frontView; }
        public void setFrontView(String frontView) { this.frontView = frontView; }

        @DynamoDBAttribute(attributeName = "RearView")
        public String getRearView() { return rearView; }
        public void setRearView(String rearView) { this.rearView = rearView; }

        @DynamoDBAttribute(attributeName = "SideView")
        public String getSideView() { return sideView; }
        public void setSideView(String sideView) { this.sideView = sideView; }

    }
}
```

A continuación, podría guardar un nuevo elemento `ProductCatalog`, con `Pictures`, tal como se muestra en el siguiente ejemplo.

```
ProductCatalogItem item = new ProductCatalogItem();

Pictures pix = new Pictures();
pix.setFrontView("http://example.com/products/123_front.jpg");
pix.setRearView("http://example.com/products/123_rear.jpg");
pix.setSideView("http://example.com/products/123_left_side.jpg");
item.setPictures(pix);

item.setId(123);

mapper.save(item);
```

El elemento `ProductCatalog` resultante tendría este aspecto (en formato JSON).

```
{
  "Id" : 123
  "Pictures" : {
    "SideView" : "http://example.com/products/123_left_side.jpg",
    "RearView" : "http://example.com/products/123_rear.jpg",
    "FrontView" : "http://example.com/products/123_front.jpg"
  }
}
```

DynamoDBHashKey

Mapea una propiedad de clase a la clave de partición de la tabla. La propiedad debe ser un escalar de tipo String, Number o Binary. La propiedad no puede ser un tipo de colección.

Supongamos que tenemos una tabla, `ProductCatalog`, cuya clave principal es `Id`. En el siguiente código Java se define una clase `CatalogItem` y se mapea su propiedad `Id` a la clave principal de la tabla `ProductCatalog` utilizando la etiqueta `@DynamoDBHashKey`.

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {
    private Integer Id;
   @DynamoDBHashKey(attributeName="Id")
   public Integer getId() {
        return Id;
   }
   public void setId(Integer Id) {
        this.Id = Id;
   }
   // Additional properties go here.
}
```

DynamoDBIgnore

Indica a la instancia `DynamoDBMapper` que la propiedad asociada debe pasarse por alto. Al guardar datos en la tabla, `DynamoDBMapper` no guarda esta propiedad en la tabla.

Se aplica al método getter o al campo de clase de una propiedad sin modelar. Si la anotación se aplica directamente al campo de clase, los métodos getter y setter correspondientes deben declararse en la misma clase.

DynamoDBIndexHashKey

Mapea una propiedad de clase a la clave de partición de un índice secundario global. La propiedad debe ser un escalar de tipo String, Number o Binary. La propiedad no puede ser un tipo de colección.

Use esta anotación si necesita `Query` un índice secundario global. Debe especificar el nombre de índice (`globalSecondaryIndexName`). Si el nombre de la propiedad de clase es distinto de la clave de partición del índice, también deberá especificar el nombre de ese atributo de índice (`attributeName`).

DynamoDBIndexRangeKey

Mapea una propiedad de clase a la clave de ordenación de un índice secundario global o un índice secundario local. La propiedad debe ser un escalar de tipo String, Number o Binary. La propiedad no puede ser un tipo de colección.

Use esta anotación si tiene que utilizar una operación `Query` en un índice secundario local o un índice secundario global y desea refinar los resultados mediante la clave de ordenación del índice. Debe especificar el nombre de índice (`globalSecondaryIndexName` o `localSecondaryIndexName`). Si el nombre de la propiedad de clase es distinto de la clave de ordenación del índice, también deberá especificar el nombre de ese atributo de índice (`attributeName`).

DynamoDBRangeKey

Mapea una propiedad de clase a la clave de ordenación de la tabla. La propiedad debe ser un escalar de tipo String, Number o Binary. No puede ser un tipo de colección.

Si la clave principal es compuesta (clave de partición y clave de ordenación), puede utilizar esta etiqueta para mapear el campo de clase a la clave de ordenación. Por ejemplo, supongamos que tenemos una tabla `Reply` en la que se almacenan las respuestas de las conversaciones de un foro. Cada conversación puede tener muchas respuestas. La clave principal de esta tabla consta de `ThreadId` y `ReplyDateTime`. `ThreadId` es la clave de partición y `ReplyDateTime` es la de orden.

En el siguiente código Java se define una clase `Reply` y se mapea a la tabla `Reply`. Se utilizan las etiquetas `@DynamoDBHashKey` y `@DynamoDBRangeKey` para identificar las propiedades de clase mapeadas a la clave principal.

```
@DynamoDBTable(tableName="Reply")
public class Reply {
    private Integer id;
    private String replyDateTime;

    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }

    @DynamoDBRangeKey(attributeName="ReplyDateTime")
    public String getReplyDateTime() { return replyDateTime; }
```

```
    public void setReplyDateTime(String replyDateTime) { this.replyDateTime =
 replyDateTime; }

   // Additional properties go here.
}
```

DynamoDBTable

Identifica la tabla de destino de DynamoDB. Por ejemplo, en el siguiente código Java se define una clase `Developer` y se mapea a la tabla `People` en DynamoDB.

```
@DynamoDBTable(tableName="People")
public class Developer { ...}
```

La anotación `@DynamoDBTable` se puede heredar. Cualquier nueva clase que herede de la clase `Developer` también se mapea a la tabla `People`. Por ejemplo, supongamos que hemos creado una clase `Lead` que hereda de la clase `Developer`. Dado que ha mapeado la clase `Developer` a la tabla `People`, los objetos de la clase `Lead` también se almacenan en la misma tabla.

La anotación `@DynamoDBTable` también se puede anular. Cualquier nueva clase que herede de la clase `Developer` de forma predeterminada se mapea a la misma tabla `People`. Sin embargo, puede anular este mapeo predeterminado. Por ejemplo, si crea una clase que hereda de la clase `Developer`, puede mapearla explícitamente a otra tabla agregando la anotación `@DynamoDBTable`, como se muestra en el siguiente ejemplo de código Java.

```
@DynamoDBTable(tableName="Managers")
public class Manager extends Developer { ...}
```

DynamoDBTypeConverted

Anotación para marcar que una propiedad usa un convertidor de tipos personalizado. Se puede usar una anotación definida por el usuario para pasar propiedades adicionales al convertidor `DynamoDBTypeConverter`.

La interfaz `DynamoDBTypeConverter` permite mapear sus propios tipos de datos arbitrarios a un tipo de datos que sea compatible de forma nativa con DynamoDB. Para obtener más información, consulte Asignación de datos arbitrarios en DynamoDB.

DynamoDBTyped

Anotación para anular el vínculo de tipo de atributo estándar. Los tipos estándar no requieren la anotación si se les aplica el vínculo de atributo predeterminado para ese tipo.

DynamoDBVersionAttribute

Identifica una propiedad de clase para almacenar un número de versión de bloqueo optimista. `DynamoDBMapper` asigna un número de versión a esta propiedad cuando guarda un elemento nuevo e incrementa su valor cada vez que se actualiza el elemento. Solo se admiten escalares de tipo Number. Para obtener más información sobre los tipos de datos, consulte Tipos de datos. Para obtener más información sobre el control de versiones, consulte DynamoDB y bloqueo positivo con el número de versión.

Ajustes de configuración opcionales para DynamoDBMapper

Al crear una instancia de `DynamoDBMapper`, presenta algunos comportamientos predeterminados que se pueden anular mediante la clase `DynamoDBMapperConfig`.

En el siguiente fragmento de código se crea una clase `DynamoDBMapper` con ajustes personalizados:

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

DynamoDBMapperConfig mapperConfig = DynamoDBMapperConfig.builder()
        .withSaveBehavior(DynamoDBMapperConfig.SaveBehavior.CLOBBER)
        .withConsistentReads(DynamoDBMapperConfig.ConsistentReads.CONSISTENT)
        .withTableNameOverride(null)

 .withPaginationLoadingStrategy(DynamoDBMapperConfig.PaginationLoadingStrategy.EAGER_LOADING)
    .build();

DynamoDBMapper mapper = new DynamoDBMapper(client, mapperConfig);
```

Para obtener más información, consulte DynamoDBMapperConfig en la Referencia de la API de AWS SDK para Java.

Puede usar los argumentos siguientes para una instancia de `DynamoDBMapperConfig`:

- Un valor de enumeración `DynamoDBMapperConfig.ConsistentReads`:
  - `EVENTUAL`: la instancia de mapeador utiliza una solicitud de lectura consistente final.

- `CONSISTENT`: la instancia de mapeador utiliza una solicitud de lectura de consistencia alta. Puede usar este ajuste opcional con las operaciones `load`, `query` o `scan` operations. Las lecturas de consistencia alta afectan al rendimiento y a la facturación; consulte la página detalles del producto de DynamoDB para obtener más información.

Si no especifica un ajuste de consistencia de lectura para la instancia de mapeador, el valor predeterminado es `EVENTUAL`.

> **Note**
>
> Este valor se aplica en las operaciones `query`, `querypage`, `load`, y `batch load` de DynamoDBMapper.

- Un valor de enumeración `DynamoDBMapperConfig.PaginationLoadingStrategy`: controla cómo la instancia de mapeador procesa una lista de datos paginados, como los resultados de una operación `query` o `scan`:
  - `LAZY_LOADING`: la instancia de mapeador carga los datos cuando es posible y conserva todos los resultados cargados en la memoria.
  - `EAGER_LOADING`: la instancia de mapeador carga los datos tan pronto como se inicializa la lista.
  - `ITERATION_ONLY`: solo se puede usar un iterador para leer datos en la lista. Durante la iteración, la lista borrará todos los resultados anteriores antes de cargar la página siguiente, para que la lista conserve como máximo una página de resultados cargados en la memoria. Por consiguiente, la lista solamente se puede recorrer en iteración una vez. Esta estrategia se recomienda para administrar elementos de gran tamaño, con el fin de reducir la sobrecarga de la memoria.

  Si no especifica una estrategia de carga de paginación para la instancia de mapeador, el valor predeterminado es `LAZY_LOADING`.
- Un valor de enumeración `DynamoDBMapperConfig.SaveBehavior`: especifica cómo la instancia de mapeador administrará los atributos durante las operaciones de almacenamiento:
  - `UPDATE`: durante una operación de almacenamiento, se actualizan todos los atributos modelados y los atributos no modelados no sufren cambios. Los tipos numéricos primitivos (byte, int, long) se establecen en 0. Los tipos de objeto se establecen en null.
  - `CLOBBER`: borra y sustituye los atributos, incluidos los no modelados, durante una operación de almacenamiento. Para ello, se elimina el elemento y se vuelve a crear. También se descartan las restricciones de los campos con versiones.

Si no especifica el comportamiento al guardar para la instancia de mapeador, el valor predeterminado es `UPDATE`.

> **Note**
>
> Las operaciones transaccionales de DynamoDBMapper no admiten la enumeración `DynamoDBMapperConfig.SaveBehavior`.

- Un objeto `DynamoDBMapperConfig.TableNameOverride`: indica a la instancia de mapeador que pase por alto el nombre de tabla especificado por la anotación `DynamoDBTable` de la clase y, en su lugar, use el nombre de tabla alternativo suministrado. Esto resulta útil cuando se particionan los datos en varias tablas en tiempo de ejecución.

Puede anular el objeto de configuración predeterminado para `DynamoDBMapper` en cada operación, conforme lo necesite.

DynamoDB y bloqueo positivo con el número de versión

El bloqueo optimista es una estrategia para asegurarse de que el elemento del lado del cliente que se va a actualizar (o eliminar) sea el mismo que figura en Amazon DynamoDB. Si utiliza esta estrategia, las escrituras en su base de datos se protegen contra posibles sobrescrituras de otros y viceversa.

Con el bloqueo optimista, cada elemento tiene un atributo que actúa como número de versión. Si recupera un elemento de una tabla, la aplicación registra el número de versión de ese elemento. Puede actualizar el elemento, pero solo si el número de versión del lado del servidor no ha cambiado. Si no hay una coincidencia de versión, significa que alguien más ha modificado el elemento antes que usted. El intento de actualización falla porque tiene una versión obsoleta del elemento. Si esto ocurre, intente recuperar el elemento y actualizarlo. El bloqueo optimista impide que sobrescriba accidentalmente los cambios realizados por otras personas. También impide que otras personas sobrescriban accidentalmente sus cambios.

Si bien puede implementar su propia estrategia de bloqueo positivo, AWS SDK para Java proporciona la anotación `@DynamoDBVersionAttribute`. En la clase de mapeo de la tabla, debe designar una propiedad en la que se almacenará el número de versión y marcarla con esta anotación. Al guardar un objeto, el elemento correspondiente de la tabla de DynamoDB tendrá un atributo en el que se almacenará el número de versión. `DynamoDBMapper` asigna un número de versión la primera vez que se guarda el objeto y aumenta automáticamente este número de versión

cada vez que se actualiza el elemento. Las solicitudes de actualización o eliminación solamente se llevan a cabo si la versión del objeto en el lado del cliente coincide con el número de versión del elemento correspondiente en la tabla de DynamoDB.

`ConditionalCheckFailedException` se lanza si:

- Utiliza el bloqueo optimista con `@DynamoDBVersionAttribute` y el valor de versión en el servidor es distinto del valor en el lado del cliente.
- Especifique sus propias limitaciones condicionales al guardar los datos utilizando `DynamoDBMapper` con `DynamoDBSaveExpression` y estas limitaciones han fallado.

> **Note**
>
> - Las tablas globales de DynamoDB usan una reconciliación del tipo "prevalece el último escritor" entre las actualizaciones simultáneas. Si usa tablas globales, prevalecerá la política del último escritor. Por tanto, en este caso, la estrategia de bloqueo no funciona según lo previsto.
> - Las operaciones de escritura transaccional `DynamoDBMapper` no admiten expresiones de anotación y condición `@DynamoDBVersionAttribute` en el mismo objeto. Si un objeto en una escritura transacciones se anota con `@DynamoDBVersionAttribute` y también tiene una expresión de condición, se producirá una SdkClientException.

Por ejemplo, en el siguiente código Java se define una clase `CatalogItem` que tiene varias propiedades. La propiedad `Version` está etiquetada con la anotación `@DynamoDBVersionAttribute`.

Example

```
@DynamoDBTable(tableName="ProductCatalog")
public class CatalogItem {

    private Integer id;
    private String title;
    private String ISBN;
    private Set<String> bookAuthors;
    private String someProp;
    private Long version;
```

```
    @DynamoDBHashKey(attributeName="Id")
    public Integer getId() { return id; }
    public void setId(Integer Id) { this.id = Id; }

    @DynamoDBAttribute(attributeName="Title")
    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    @DynamoDBAttribute(attributeName="ISBN")
    public String getISBN() { return ISBN; }
    public void setISBN(String ISBN) { this.ISBN = ISBN;}

    @DynamoDBAttribute(attributeName = "Authors")
    public Set<String> getBookAuthors() { return bookAuthors; }
    public void setBookAuthors(Set<String> bookAuthors) { this.bookAuthors =
 bookAuthors; }

    @DynamoDBIgnore
    public String getSomeProp() { return someProp;}
    public void setSomeProp(String someProp) {this.someProp = someProp;}

    @DynamoDBVersionAttribute
    public Long getVersion() { return version; }
    public void setVersion(Long version) { this.version = version;}
}
```

Puede aplicar la anotación `@DynamoDBVersionAttribute` a los tipos que admiten valores null; estos están disponibles en las clases encapsuladoras primitivas que proporcionan un tipo que admite valores null, tales como `Long` e `Integer`.

El bloqueo optimista afecta a los siguientes métodos de `DynamoDBMapper` como se indica a continuación:

- `save`: para un elemento nuevo, `DynamoDBMapper` asigna un número de versión inicial de 1. Si recupera un elemento, actualiza una o varias de sus propiedades e intenta guardar los cambios, la operación de almacenamiento solamente se lleva a cabo si el número de versión del lado del cliente coincide con el número de versión del lado del servidor. `DynamoDBMapper` incrementa el número de versión automáticamente.

- `delete`: el método `delete` toma un objeto como parámetro y `DynamoDBMapper` lleva a cabo una comprobación de versión antes de eliminar el elemento. La comprobación de versión se puede deshabilitar si se especifica `DynamoDBMapperConfig.SaveBehavior.CLOBBER` en la solicitud.

  La implementación interna del bloqueo optimista en `DynamoDBMapper` utiliza la compatibilidad con las acciones de actualización condicional y eliminación condicional que DynamoDB proporciona.

- `transactionWrite` —
  - `Put`: para un elemento nuevo, `DynamoDBMapper` asigna un número de versión inicial de 1. Si recupera un elemento, actualiza una o varias de sus propiedades e intenta guardar los cambios, la operación put solamente se lleva a cabo si el número de versión del lado del cliente coincide con el número de versión del lado del servidor. `DynamoDBMapper` incrementa el número de versión automáticamente.
  - `Update`: para un elemento nuevo, `DynamoDBMapper` asigna un número de versión inicial de 1. Si recupera un elemento, actualiza una o varias de sus propiedades e intenta guardar los cambios, la operación update solamente se lleva a cabo si el número de versión del lado del cliente coincide con el número de versión del lado del servidor. `DynamoDBMapper` incrementa el número de versión automáticamente.
  - `Delete`: `DynamoDBMapper` realiza una comprobación de versión antes de eliminar el elemento. La operación de eliminación solo se realiza correctamente si coincide el número de versión en el lado del cliente y en el lado del servidor.
  - `ConditionCheck`: la anotación `@DynamoDBVersionAttribute` no es compatible con `ConditionCheck`. Se producirá una excepción SdkClientException cuando un elemento `ConditionCheck` se anote con `@DynamoDBVersionAttribute`.

Deshabilitación del bloqueo positivo

Para deshabilitar el bloqueo optimista, puede cambiar el valor de enumeración `DynamoDBMapperConfig.SaveBehavior` de `UPDATE` a `CLOBBER`. Para ello, puede crear una instancia de `DynamoDBMapperConfig` que omita la comprobación de versión y usar esta instancia en todas las solicitudes. Para obtener información acerca de `DynamoDBMapperConfig.SaveBehavior` y otros parámetros opcionales de `DynamoDBMapper`, consulte Ajustes de configuración opcionales para DynamoDBMapper .

También puede establecer el comportamiento de bloqueo para una operación específica. Por ejemplo, en el siguiente fragmento de código Java se usa `DynamoDBMapper` para guardar un

elemento de catálogo. Se agrega el parámetro opcional `DynamoDBMapperConfig.SaveBehavior` al método `DynamoDBMapperConfig` para especificar `save`.

> **Note**
>
> El método transactionWrite method no admite la configuración DynamoDBMapperConfig.SaveBehavior. No se admite la deshabilitación del bloqueo optimista para transactionWrite.

Example

```
DynamoDBMapper mapper = new DynamoDBMapper(client);

// Load a catalog item.
CatalogItem item = mapper.load(CatalogItem.class, 101);
item.setTitle("This is a new title for the item");
...
// Save the item.
mapper.save(item,
    new DynamoDBMapperConfig(
        DynamoDBMapperConfig.SaveBehavior.CLOBBER));
```

Asignación de datos arbitrarios en DynamoDB

Además de los tipos de Java admitidos (consulte Tipos de datos compatibles con asignador DynamoDBMapper para Java), puede utilizar tipos de la aplicación para los cuales no exista un mapeo directo a los tipos de Amazon DynamoDB. Para asignar estos tipos, debe proporcionar una implementación que convierta el tipo complejo en un tipo admitido por DynamoDB y viceversa, y anotar el método de acceso al tipo complejo mediante la anotación `@DynamoDBTypeConverted`. El código convertidor transforma los datos al guardar o cargar los objetos. También se usa para todas las operaciones que consumen tipos complejos. Tenga en cuenta que, al comparar datos durante las operaciones de consulta y examen, las comparaciones se realizan respecto a los datos almacenados en DynamoDB.

Por ejemplo, tomemos la siguiente clase `CatalogItem` que define una propiedad, `Dimension`, del tipo `DimensionType`. Esta propiedad almacena las dimensiones del elemento, tales como altura, anchura o espesor. Supongamos que decide almacenar estas dimensiones del elemento en una cadena (por ejemplo, 8,5x11x0,05) en DynamoDB. En el ejemplo siguiente se proporciona el

código convertidor que convierte el objeto `DimensionType` en una cadena y una cadena al tipo `DimensionType`.

> **Note**
>
> En este ejemplo de código se supone que ya ha cargado datos en DynamoDB para su cuenta siguiendo las instrucciones de la sección Creación de tablas y carga de datos para ejemplos de código en DynamoDB.
> Para obtener instrucciones paso a paso acerca de cómo ejecutar el siguiente ejemplo, consulte Ejemplos de código Java.

Example

```
public class DynamoDBMapperExample {

    static AmazonDynamoDB client;

    public static void main(String[] args) throws IOException {

        // Set the AWS region you want to access.
        Regions usWest2 = Regions.US_WEST_2;
        client = AmazonDynamoDBClientBuilder.standard().withRegion(usWest2).build();

        DimensionType dimType = new DimensionType();
        dimType.setHeight("8.00");
        dimType.setLength("11.0");
        dimType.setThickness("1.0");

        Book book = new Book();
        book.setId(502);
        book.setTitle("Book 502");
        book.setISBN("555-5555555555");
        book.setBookAuthors(new HashSet<String>(Arrays.asList("Author1", "Author2")));
        book.setDimensions(dimType);

        DynamoDBMapper mapper = new DynamoDBMapper(client);
        mapper.save(book);

        Book bookRetrieved = mapper.load(Book.class, 502);
        System.out.println("Book info: " + "\n" + bookRetrieved);
```

```
        bookRetrieved.getDimensions().setHeight("9.0");
        bookRetrieved.getDimensions().setLength("12.0");
        bookRetrieved.getDimensions().setThickness("2.0");

        mapper.save(bookRetrieved);

        bookRetrieved = mapper.load(Book.class, 502);
        System.out.println("Updated book info: " + "\n" + bookRetrieved);
    }

    @DynamoDBTable(tableName = "ProductCatalog")
    public static class Book {
        private int id;
        private String title;
        private String ISBN;
        private Set<String> bookAuthors;
        private DimensionType dimensionType;

        // Partition key
        @DynamoDBHashKey(attributeName = "Id")
        public int getId() {
            return id;
        }

        public void setId(int id) {
            this.id = id;
        }

        @DynamoDBAttribute(attributeName = "Title")
        public String getTitle() {
            return title;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        @DynamoDBAttribute(attributeName = "ISBN")
        public String getISBN() {
            return ISBN;
        }

        public void setISBN(String ISBN) {
```

```
            this.ISBN = ISBN;
        }

        @DynamoDBAttribute(attributeName = "Authors")
        public Set<String> getBookAuthors() {
            return bookAuthors;
        }

        public void setBookAuthors(Set<String> bookAuthors) {
            this.bookAuthors = bookAuthors;
        }

        @DynamoDBTypeConverted(converter = DimensionTypeConverter.class)
        @DynamoDBAttribute(attributeName = "Dimensions")
        public DimensionType getDimensions() {
            return dimensionType;
        }

        @DynamoDBAttribute(attributeName = "Dimensions")
        public void setDimensions(DimensionType dimensionType) {
            this.dimensionType = dimensionType;
        }

        @Override
        public String toString() {
            return "Book [ISBN=" + ISBN + ", bookAuthors=" + bookAuthors + ",
dimensionType= "
                    + dimensionType.getHeight() + " X " + dimensionType.getLength() + "
X "
                    + dimensionType.getThickness()
                    + ", Id=" + id + ", Title=" + title + "]";
        }
    }

    static public class DimensionType {

        private String length;
        private String height;
        private String thickness;

        public String getLength() {
            return length;
        }
```

```
        public void setLength(String length) {
            this.length = length;
        }

        public String getHeight() {
            return height;
        }

        public void setHeight(String height) {
            this.height = height;
        }

        public String getThickness() {
            return thickness;
        }

        public void setThickness(String thickness) {
            this.thickness = thickness;
        }
    }

    // Converts the complex type DimensionType to a string and vice-versa.
    static public class DimensionTypeConverter implements DynamoDBTypeConverter<String,
DimensionType> {

        @Override
        public String convert(DimensionType object) {
            DimensionType itemDimensions = (DimensionType) object;
            String dimension = null;
            try {
                if (itemDimensions != null) {
                    dimension = String.format("%s x %s x %s",
itemDimensions.getLength(), itemDimensions.getHeight(),
                            itemDimensions.getThickness());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            return dimension;
        }

        @Override
        public DimensionType unconvert(String s) {
```

```
            DimensionType itemDimension = new DimensionType();
            try {
                if (s != null && s.length() != 0) {
                    String[] data = s.split("x");
                    itemDimension.setLength(data[0].trim());
                    itemDimension.setHeight(data[1].trim());
                    itemDimension.setThickness(data[2].trim());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }

            return itemDimension;
        }
    }
}
```

Ejemplos de DynamoDBMapper

El AWS SDK proporciona la clase `DynamoDBMapper`, que le permite asignar las clases del cliente a las tablas de DynamoDB. Para usar `DynamoDBMapper`, se define la relación entre los elementos de una tabla de DynamoDB y sus instancias de objetos correspondientes en el código. La clase `DynamoDBMapper` permite realizar varias operaciones de creación, lectura, actualización y eliminación (CRUD, Create, Read, Update y Delete) en elementos y ejecutar consultas y análisis en tablas.

Para obtener más información sobre cómo usar `DynamoDBMapper`, consulte DynamoDB Examples Using the AWS SDK for Java en AWS SDK for Java 1.x Developer Guide.

Java 2.x: cliente mejorado de DynamoDB

El cliente mejorado de DynamoDB es una biblioteca de alto nivel que forma parte de AWS SDK para Java versión 2 (v2). Ofrece una forma sencilla de asignar clases del cliente a tablas de DynamoDB. Puede definir las relaciones entre las tablas y sus correspondientes clases de modelo en el código. Después de definir estas relaciones, puede realizar intuitivamente varias operaciones de creación, lectura, actualización o eliminación (CRUD) en tablas o elementos en DynamoDB.

Para obtener más información sobre cómo puede utilizar el cliente mejorado con DynamoDB, consulte Uso del cliente mejorado de DynamoDB en AWS SDK para Java versión 2.x .

Uso del modelo de documento de .NET en DynamoDB

El AWS SDK para .NET proporciona clases de modelo de documento que encapsulan algunas de las operaciones de bajo nivel de Amazon DynamoDB y así le ayudan a simplificar la codificación. En el modelo de documento, las clases principales son `Table` y `Document`. La clase `Table` proporciona métodos de operaciones de datos, como `PutItem`, `GetItem` y `DeleteItem`. Además, proporciona los métodos `Query` y `Scan`. La clase `Document` representa un solo elemento de una tabla.

Las clases del modelo de documento citadas están disponibles en el espacio de nombres `Amazon.DynamoDBv2.DocumentModel`.

> **Note**
>
> Las clases del modelo de documento no se pueden usar para crear, actualizar ni eliminar tablas. Sin embargo, el modelo de documento admite la mayoría de las operaciones de datos habituales.

Temas

- Tipos de datos compatibles

Tipos de datos compatibles

El modelo de documento admite un conjunto de tipos de datos de .NET primitivos y tipos de datos de colecciones. El modelo admite los siguientes tipos de datos primitivos.

- `bool`
- `byte`
- `char`
- `DateTime`
- `decimal`
- `double`
- `float`
- `Guid`
- `Int16`
- `Int32`

- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

En la tabla siguiente se resume el mapeo de los tipos de .NET anteriores a los tipos de DynamoDB.

Tipo de .NET primitivo	Tipo DynamoDB
Todos los tipos de números	`N` (tipo Number)
Todos los tipos de cadenas	`S` (tipo String)
MemoryStream, byte[]	`B` (tipo Binary)
bool	`N` (tipo Number), 0 representa false (falso) y 1 representa true (verdadero).
DateTime	`S` (tipo String). Los valores `DateTime` se almacenan como cadenas con formato ISO-8601.
Guid	`S` (tipo String).
Tipos de recopilación (List, HashSet y array)	`BS` (tipo Binary Set), `SS` (tipo String Set) y `NS` (tipo Number Set)

AWS SDK para .NET define tipos para establecer una correspondencia entre tipos booleanos, nulos, listas y mapas de DynamoDB y la API de modelo de documento de .NET:

- Utilice `DynamoDBBool` para tipo booleano.
- Utilice `DynamoDBNull` para tipo nulo.
- Utilice `DynamoDBList` para tipo lista.
- Utilice `Document` para tipo mapa.

> **Note**
>
> - Se admiten valores binarios vacíos.
> - Se admite la lectura de valores de cadena vacíos. Los valores de atributo de cadena vacíos se admiten dentro de los valores de atributo del tipo de conjunto de cadenas mientras se escribe en DynamoDB. Los valores de atributo de cadena vacíos del tipo de cadena y los valores de cadena vacíos contenidos en el tipo lista o mapa se eliminan de las solicitudes de escritura

Uso del modelo de persistencia de objetos de .NET y DynamoDB

El AWS SDK para .NET proporciona un modelo de persistencia de objetos que le permite mapear las clases en el cliente a una tabla de Amazon DynamoDB. A continuación, cada instancia de objeto se mapea a un elemento en las tablas correspondientes. Para guardar los objetos del lado del cliente en las tablas, el modelo de persistencia de objetos proporciona la clase `DynamoDBContext`, un punto de entrada a DynamoDB. Esta categoría le ofrece una conexión a DynamoDB y le permite obtener acceso a tablas, realizar diversas operaciones CRUD y ejecutar consultas.

El modelo de persistencia de objetos proporciona un conjunto de atributos para mapear las clases del lado del cliente a tablas, y las propiedades/campos a atributos de tabla.

> **Note**
>
> El modelo de persistencia de objetos no proporciona una API para crear, actualizar o eliminar tablas. Solo ofrece operaciones de datos. Puede usar la API de bajo nivel AWS SDK para .NET para crear, actualizar y eliminar tablas.

En el ejemplo siguiente se muestra cómo funciona el modelo de persistencia de objetos. Comienza con la tabla `ProductCatalog`. Su clave principal es `Id`.

```
ProductCatalog(Id, ...)
```

Suponga que tiene una clase `Book` con las propiedades `Title`, `ISBN` y `Authors`. Puede mapear la clase `Book` a la tabla `ProductCatalog` agregando los atributos definidos por el modelo de persistencia de objetos, como se muestra en el siguiente ejemplo de código C#.

Example

```
[DynamoDBTable("ProductCatalog")]
  public class Book
  {
    [DynamoDBHashKey]
    public int Id { get; set; }

    public string Title { get; set; }
    public int ISBN { get; set; }

    [DynamoDBProperty("Authors")]
    public List<string> BookAuthors { get; set; }

    [DynamoDBIgnore]
    public string CoverPage { get; set; }
  }
```

En el ejemplo anterior, el atributo `DynamoDBTable` mapea la clase `Book` a la tabla `ProductCatalog`.

El modelo de persistencia de objetos admite el mapeo tanto explícito como predeterminado entre las propiedades de clase y los atributos de tabla.

- Mapeo explícito: para mapear una propiedad a una clave principal, debe usar los atributos `DynamoDBHashKey` y `DynamoDBRangeKey` del modelo de persistencia de objetos. Además, en el caso de los atributos sin clave principal, si un nombre de propiedad de la clase y el atributo de tabla correspondiente al que desea mapearlo no son iguales, debe definir el mapeo agregando explícitamente el atributo `DynamoDBProperty`.

  En el ejemplo anterior, la propiedad `Id` se mapea a la clave principal con el mismo nombre y la propiedad `BookAuthors` se mapea al atributo `Authors` de la tabla `ProductCatalog`.
- Mapeo predeterminado: de forma predeterminada, el modelo de persistencia de objetos mapea las propiedades de clase a los atributos con el mismo nombre de la tabla.

  En el ejemplo anterior, las propiedades `Title` e `ISBN` se mapean a los atributos del mismo nombre de la tabla `ProductCatalog`.

No tiene que mapear cada propiedad de clase. Puede identificar estas propiedades agregando el atributo `DynamoDBIgnore`. Al guardar una instancia de `Book` en la tabla, `DynamoDBContext` no

incluye la propiedad `CoverPage`. Tampoco se devolverá esta propiedad cuando se recupere la instancia del libro.

Puede mapear propiedades de tipos primitivos de .NET, como int o string. También puede mapear cualquier tipo de datos arbitrarios, siempre y cuando proporcione un convertidor adecuado para mapear los datos arbitrarios a uno de los tipos de DynamoDB. Para obtener más información sobre cómo mapear tipos arbitrarios, consulte Mapeo de datos arbitrarios con DynamoDB mediante el modelo de persistencia de objetos de AWS SDK para .NET.

El modelo de persistencia de objetos admite el bloqueo optimista. Durante una operación de actualización, esto garantiza que se disponga de la última copia del elemento que se va a actualizar. Para obtener más información, consulte Bloqueo positivo mediante DynamoDB y el modelo de persistencia de objetos de AWS SDK para .NET.

Para obtener más información, consulte los siguientes temas.

Temas

- Tipos de datos compatibles
- Atributos de DynamoDB del modelo de persistencia de objetos de .NET
- Clase DynamoDBContext del modelo de persistencia de objetos de .NET
- Bloqueo positivo mediante DynamoDB y el modelo de persistencia de objetos de AWS SDK para .NET
- Mapeo de datos arbitrarios con DynamoDB mediante el modelo de persistencia de objetos de AWS SDK para .NET

Tipos de datos compatibles

El modelo de persistencia de objetos admite un conjunto de tipos de datos, colecciones y tipos de datos arbitrarios de .NET primitivos. El modelo admite los siguientes tipos de datos primitivos.

- `bool`
- `byte`
- `char`
- `DateTime`
- `decimal`
- `double`

- `float`
- `Int16`
- `Int32`
- `Int64`
- `SByte`
- `string`
- `UInt16`
- `UInt32`
- `UInt64`

El modelo de persistencia de objetos también admite los tipos de colección de .NET. `DynamoDBContext` puede convertir tipos de colección concretos y objetos CLR estándar (POCO, por sus siglas en inglés) simples.

En la tabla siguiente se resume el mapeo de los tipos de .NET anteriores a los tipos de DynamoDB.

Tipo de .NET primitivo	Tipo DynamoDB
Todos los tipos de números	`N` (tipo Number)
Todos los tipos de cadenas	`S` (tipo String)
MemoryStream, byte[]	`B` (tipo Binary)
bool	`N` (tipo Number), 0 representa false (falso) y 1 representa true (verdadero).
Tipos de colección	`BS` (tipo Binary Set), `SS` (tipo String Set) y `NS` (tipo Number Set)
DateTime	`S` (tipo String). Los valores `DateTime` se almacenan como cadenas con formato ISO-8601.

El modelo de persistencia de objetos también admite los tipos de datos arbitrarios. Sin embargo, debe proporcionar el código de convertidor para mapear los tipos complejos a los tipos de DynamoDB.

> **Note**
>
> - Se admiten valores binarios vacíos.
> - Se admite la lectura de valores de cadena vacíos. Los valores de atributo de cadena vacíos se admiten dentro de los valores de atributo del tipo de conjunto de cadenas mientras se escribe en DynamoDB. Los valores de atributo de cadena vacíos del tipo de cadena y los valores de cadena vacíos contenidos en el tipo lista o mapa se eliminan de las solicitudes de escritura

Atributos de DynamoDB del modelo de persistencia de objetos de .NET

En esta sección se describen los atributos que ofrece el modelo de persistencia de objetos para que pueda mapear las clases y propiedades a tablas y atributos de DynamoDB.

> **Note**
>
> En los atributos siguientes, solo son obligatorios `DynamoDBTable` y `DynamoDBHashKey`.

DynamoDBGlobalSecondaryIndexHashKey

Mapea una propiedad de clase a la clave de partición de un índice secundario global. Use este atributo si necesita `Query` un índice secundario global.

DynamoDBGlobalSecondaryIndexRangeKey

Mapea una propiedad de clase a la clave de ordenación de un índice secundario global. Use este atributo si tiene que utilizar una operación `Query` en un índice secundario global y desea refinar los resultados mediante la clave de ordenación del índice.

DynamoDBHashKey

Mapea una propiedad de clase a la clave de partición de la clave principal de la tabla. Los atributos de clave principal no pueden ser un tipo de colección.

En los siguientes ejemplos de código C# se mapea la clase `Book` a la tabla `ProductCatalog` y la propiedad `Id` a la clave de partición de la clave principal de la tabla.

```
[DynamoDBTable("ProductCatalog")]
public class Book
{
    [DynamoDBHashKey]
    public int Id { get; set; }

    // Additional properties go here.
}
```

DynamoDBIgnore

Indica que la propiedad asociada debe pasarse por alto. Si no desea guardar ninguna de las propiedades de clase, puede agregar este atributo para indicar a `DynamoDBContext` que no incluya esta propiedad cuando guarde objetos en la tabla.

DynamoDBLocalSecondaryIndexRangeKey

Mapea una propiedad de clase a la clave de ordenación de un índice secundario local. Use este atributo si tiene que utilizar una operación `Query` en un índice secundario local y desea refinar los resultados mediante la clave de ordenación del índice.

DynamoDBProperty

Mapea una propiedad de clase a un atributo de tabla. Si la propiedad de clase se mapea al atributo de tabla con el mismo nombre, no es preciso especificarlo. Sin embargo, si los nombres no son iguales, puede utilizar esta etiqueta para realizar el mapeo. En la siguiente instrucción de C#, `DynamoDBProperty` mapea la propiedad `BookAuthors` al atributo `Authors` de la tabla.

```
[DynamoDBProperty("Authors")]
public List<string> BookAuthors { get; set; }
```

`DynamoDBContext` utiliza esta información de mapeo para crear el atributo `Authors` al guardar datos de objetos en la tabla correspondiente.

DynamoDBRenamable

Especifica un nombre alternativo para una propiedad de clase. Esto resulta útil si va a escribir un convertidor personalizado para mapear datos arbitrarios a una tabla de DynamoDB cuando el nombre de una propiedad de clase sea distinto del nombre del atributo de tabla.

DynamoDBRangeKey

Mapea una propiedad de clase a la clave de ordenación de la clave principal de la tabla. Si la tabla tiene una clave principal compuesta (clave de partición y clave de ordenación), entonces debe especificar ambos atributos, `DynamoDBHashKey` y `DynamoDBRangeKey`, en el mapeo de clase.

Por ejemplo, en el ejemplo de tabla `Reply`, la clave principal consta de la clave de partición `Id` y de la clave de ordenación `Replenishment`. En el siguiente ejemplo de código C# se mapea la clase `Reply` a la tabla `Reply`. La definición de clase también indica que dos de sus propiedades se mapean a la clave principal.

```
[DynamoDBTable("Reply")]
public class Reply
{
   [DynamoDBHashKey]
   public int ThreadId { get; set; }
   [DynamoDBRangeKey]
   public string Replenishment { get; set; }

   // Additional properties go here.
}
```

DynamoDBTable

Identifica la tabla de destino de DynamoDB a la que se mapea la clase. Por ejemplo, en el siguiente ejemplo de código C# se mapea la clase `Developer` a la tabla `People` de DynamoDB.

```
[DynamoDBTable("People")]
public class Developer { ...}
```

Este atributo se puede heredar o anular.

- El atributo `DynamoDBTable` se puede heredar. En el ejemplo anterior, si agrega una nueva clase, `Lead`, que hereda de la clase `Developer`, también se mapea a la tabla `People`. Ambos objetos, `Developer` y `Lead`, se almacenan en la tabla `People`.
- El atributo `DynamoDBTable` también se puede anular. En el siguiente ejemplo de código C#, la clase `Manager` hereda de la clase `Developer`. Sin embargo, la adición explícita del atributo `DynamoDBTable` mapea la clase a otra tabla (`Managers`).

  ```
  [DynamoDBTable("Managers")]
  ```

```
public class Manager : Developer { ...}
```

Puede agregar el parámetro opcional, `LowerCamelCaseProperties`, para solicitar que DynamoDB cambie a minúscula la primera letra del nombre de la propiedad cuando almacene los objetos en una tabla, como se muestra en el siguiente fragmento de código C#.

```
[DynamoDBTable("People", LowerCamelCaseProperties=true)]
public class Developer
{
    string DeveloperName;
    ...
}
```

Al guardar instancias de la clase `Developer`, `DynamoDBContext` guarda la propiedad `DeveloperName` como `developerName`.

DynamoDBVersion

Identifica una propiedad de clase para almacenar el número de versión del elemento. Para obtener más información sobre el control de versiones, consulte Bloqueo positivo mediante DynamoDB y el modelo de persistencia de objetos de AWS SDK para .NET.

Clase DynamoDBContext del modelo de persistencia de objetos de .NET

La clase `DynamoDBContext` es el punto de entrada de la base de datos de Amazon DynamoDB. Proporciona conexión con DynamoDB y le permite acceder a los datos de diversas tablas, realizar distintas operaciones CRUD y ejecutar consultas. La clase `DynamoDBContext` proporciona los métodos siguientes.

Temas

- CreateMultiTableBatchGet
- CreateMultiTableBatchWrite
- CreateBatchGet
- CreateBatchWrite
- Eliminar
- Dispose
- ExecuteBatchGet

- ExecuteBatchWrite
- FromDocument
- FromQuery
- FromScan
- GetTargetTable
- Cargar
- Consultar
- Save (Guardar)
- Examen
- ToDocument
- Especificación de parámetros opcionales para DynamoDBContext

CreateMultiTableBatchGet

Crea un objeto `MultiTableBatchGet`, que consta de varios objetos `BatchGet` individuales. Cada uno de estos objetos `BatchGet` se puede usar para recuperar elementos de una sola tabla de DynamoDB.

Para recuperar elementos de una o varias tablas, utilice el método `ExecuteBatchGet` y pase el objeto `MultiTableBatchGet` como parámetro.

CreateMultiTableBatchWrite

Crea un objeto `MultiTableBatchWrite`, que consta de varios objetos `BatchWrite` individuales. Cada uno de estos objetos `BatchWrite` se puede usar para escribir o eliminar elementos de una sola tabla de DynamoDB.

Para escribir en una o varias tablas, utilice el método `ExecuteBatchWrite` y pase el objeto `MultiTableBatchWrite` como parámetro.

CreateBatchGet

Crea un objeto `BatchGet` que puede usar para recuperar varios elementos de una tabla.

CreateBatchWrite

Crea un objeto `BatchWrite` que puede usar para colocar o eliminar varios elementos en una tabla.

Eliminar

Elimina un elemento de la tabla. El método requiere la clave principal del elemento que se desea eliminar. Como parámetro de este método, puede proporcionar el valor de la clave principal o un objeto del lado del cliente que contenga un valor de clave .

- Si especifica un objeto del lado del cliente como parámetro y ha habilitado el bloqueo optimista, la eliminación se llevará a cabo correctamente solo si las versiones del lado del cliente y del lado del servidor del objeto coinciden.
- Si especifica únicamente el valor de clave principal como parámetro, la eliminación se llevará a cabo correctamente tanto si ha habilitado bloqueo optimista como si no.

> **Note**
>
> Para ejecutar esta operación en segundo plano, use el método `DeleteAsync` en su lugar.

Dispose

Elimina todos los recursos administrados y no administrados.

ExecuteBatchGet

Lee datos en una o varias tablas y procesa todos los objetos `BatchGet` de un objeto `MultiTableBatchGet`.

> **Note**
>
> Para ejecutar esta operación en segundo plano, use el método `ExecuteBatchGetAsync` en su lugar.

ExecuteBatchWrite

Escribe o elimina datos en una o varias tablas y procesa todos los objetos `BatchWrite` de un objeto `MultiTableBatchWrite`.

> **Note**
>
> Para ejecutar esta operación en segundo plano, use el método `ExecuteBatchWriteAsync` en su lugar.

FromDocument

Dada una instancia de un objeto `Document`, el método `FromDocument` devuelve una instancia de una clase del lado del cliente.

Esto resulta útil si desea utilizar las clases del modelo de documento junto con el modelo de persistencia de objetos para realizar operaciones con datos. Para obtener más información sobre las clases del modelo de documentos que se proporcionan en AWS SDK para .NET, consulte Uso del modelo de documento de .NET en DynamoDB.

Supongamos que tiene un objeto `Document` denominado `doc` que contiene una representación de un elemento `Forum`. (Para saber cómo construir este objeto, consulte la descripción del método `ToDocument` más adelante en este tema). Puede utilizar `FromDocument` para recuperar el elemento `Forum` del `Document` como se muestra en el siguiente ejemplo de código C#.

Example

```
forum101 = context.FromDocument<Forum>(101);
```

> **Note**
>
> Si el objeto `Document` implementa la interfaz `IEnumerable`, puede usar el método `FromDocuments` en su lugar. Esto le permitirá recorrer en iteración todas las instancias de la clase contenidas en `Document`.

FromQuery

Ejecuta una operación `Query` con los parámetros de la consulta definidos en un objeto `QueryOperationConfig`.

> **Note**
>
> Para ejecutar esta operación en segundo plano, use el método `FromQueryAsync` en su lugar.

FromScan

Ejecuta una operación `Scan` con los parámetros del análisis definidos en un objeto `ScanOperationConfig`.

> **Note**
>
> Para ejecutar esta operación en segundo plano, use el método `FromScanAsync` en su lugar.

GetTargetTable

Recupera la tabla de destino del tipo especificado. Esto resulta útil si va a escribir un convertidor personalizado para mapear datos arbitrarios a una tabla de DynamoDB y tiene que determinar qué tabla está asociada con un tipo de datos personalizado.

Cargar

Recupera un elemento de una tabla. El método requiere solo la clave principal del elemento que se desea recuperar.

De forma predeterminada, DynamoDB devuelve el elemento con valores que presentan consistencia final. Para obtener más información sobre el modelo de consistencia final, consulte Coherencia de lectura de DynamoDB.

El método `Load` o `LoadAsync` llama a la operación GetItem, que requiere que especifique la clave principal de la tabla. Como `GetItem` omite el parámetro `IndexName`, no puede cargar un elemento con la partición o la clave de clasificación de un índice. Por lo tanto, debe utilizar la clave principal de la tabla para cargar un elemento.

> **Note**
>
> Para ejecutar esta operación en segundo plano, use el método `LoadAsync` en su lugar. Para ver un ejemplo del uso del método `LoadAsync` para realizar operaciones CRUD de alto nivel en una tabla de DynamoDB, consulte el siguiente ejemplo.

```
/// <summary>
/// Shows how to perform high-level CRUD operations on an Amazon DynamoDB
/// table.
/// </summary>
public class HighLevelItemCrud
{
    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();
        DynamoDBContext context = new DynamoDBContext(client);
        await PerformCRUDOperations(context);
    }

    public static async Task PerformCRUDOperations(IDynamoDBContext context)
    {
        int bookId = 1001; // Some unique value.
        Book myBook = new Book
        {
            Id = bookId,
            Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
            Isbn = "111-1111111001",
            BookAuthors = new List<string> { "Author 1", "Author 2" },
        };

        // Save the book to the ProductCatalog table.
        await context.SaveAsync(myBook);

        // Retrieve the book from the ProductCatalog table.
        Book bookRetrieved = await context.LoadAsync<Book>(bookId);

        // Update some properties.
        bookRetrieved.Isbn = "222-2222221001";

        // Update existing authors list with the following values.
```

```
            bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author x" };
            await context.SaveAsync(bookRetrieved);

            // Retrieve the updated book. This time, add the optional
            // ConsistentRead parameter using DynamoDBContextConfig object.
            await context.LoadAsync<Book>(bookId, new DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            // Delete the book.
            await context.DeleteAsync<Book>(bookId);

            // Try to retrieve deleted book. It should return null.
            Book deletedBook = await context.LoadAsync<Book>(bookId, new
DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            if (deletedBook == null)
            {
                Console.WriteLine("Book is deleted");
            }
        }
    }
```

Consultar

Consulta una tabla basándose en los parámetros de consulta que haya proporcionado.

Solo se puede consultar una tabla si cuenta con una clave principal compuesta (una clave de partición y una clave de ordenación). Al realizar la consulta, debe especificar una clave de partición y una condición que se aplica a la clave de ordenación.

Supongamos que tenemos una clase `Reply` del lado del cliente mapeada a la tabla `Reply` de DynamoDB. En el siguiente ejemplo de código C# se consulta la tabla `Reply` para buscar las respuestas de las conversaciones de un foro publicadas en los últimos 15 días. La tabla `Reply` tiene una clave principal cuya clave de partición es `Id` y cuya clave de ordenación es `ReplyDateTime`.

Example

```
DynamoDBContext context = new DynamoDBContext(client);

string replyId = "DynamoDB#DynamoDB Thread 1"; //Partition key
DateTime twoWeeksAgoDate = DateTime.UtcNow.Subtract(new TimeSpan(14, 0, 0, 0)); // Date
 to compare.
IEnumerable<Reply> latestReplies = context.Query<Reply>(replyId,
 QueryOperator.GreaterThan, twoWeeksAgoDate);
```

Devuelve una colección de objetos `Reply`.

El método `Query` devuelve una colección `IEnumerable` de "carga diferida". Inicialmente devuelve una sola página de resultados y, a continuación, realiza una llamada de servicio para obtener la página siguiente si es necesario. Para obtener todos los elementos coincidentes, solo tiene que recorrer en iteración el `IEnumerable`.

Si la tabla tiene una clave principal simple (clave de partición), no puede usar el método `Query`. En su lugar, puede usar el método `Load` y proporcionar la clave de partición para recuperar el elemento.

> **Note**
>
> Para ejecutar esta operación en segundo plano, use el método `QueryAsync` en su lugar.

Save (Guardar)

Guarda el objeto especificado en la tabla. Si la clave principal especificada en el objeto de entrada no existe en la tabla, el método agrega un nuevo elemento a la tabla. Si la clave principal sí está presente, el método actualiza el elemento.

Si ha configurado el bloqueo optimista, la actualización solo se llevará a cabo correctamente si las versiones del elemento del lado del cliente y del lado del servidor coinciden. Para obtener más información, consulte Bloqueo positivo mediante DynamoDB y el modelo de persistencia de objetos de AWS SDK para .NET.

> **Note**
>
> Para ejecutar esta operación en segundo plano, use el método `SaveAsync` en su lugar.

Examen

Realiza un examen de toda la tabla.

Puede filtrar el resultado del examen especificando una condición de examen. La condición se puede evaluar según cualesquiera atributos de la tabla. Supongamos que tenemos una clase `Book` del lado del cliente mapeada a la tabla `ProductCatalog` de DynamoDB. En el siguiente ejemplo de código C# se examina la tabla y se devuelven solamente aquellos elementos de libro cuyo precio es menor que 0.

Example

```
IEnumerable<Book> itemsWithWrongPrice = context.Scan<Book>(
                    new ScanCondition("Price", ScanOperator.LessThan, price),
                    new ScanCondition("ProductCategory", ScanOperator.Equal, "Book")
      );
```

El método `Scan` devuelve una colección `IEnumerable` de "carga diferida". Inicialmente devuelve una sola página de resultados y, a continuación, realiza una llamada de servicio para obtener la página siguiente si es necesario. Para obtener todos los elementos coincidentes, solo tiene que recorrer en iteración la colección `IEnumerable`.

Por motivos de desempeño, debe consultar las tablas y evitar examinarlas.

> **Note**
>
> Para ejecutar esta operación en segundo plano, use el método `ScanAsync` en su lugar.

ToDocument

Devuelve una instancia de la clase `Document` del modelo de documento de la instancia de clase.

Esto resulta útil si desea utilizar las clases del modelo de documento junto con el modelo de persistencia de objetos para realizar operaciones con datos. Para obtener más información sobre las clases del modelo de documentos que se proporcionan en AWS SDK para .NET, consulte Uso del modelo de documento de .NET en DynamoDB.

Supongamos que tenemos una clase del lado del cliente mapeada al ejemplo de tabla `Forum`. Puede usar una `DynamoDBContext` para obtener un elemento, como un objeto `Document`, de la tabla `Forum`, como se muestra en el siguiente ejemplo de código C#.

Example

```
DynamoDBContext context = new DynamoDBContext(client);

Forum forum101 = context.Load<Forum>(101); // Retrieve a forum by primary key.
Document doc = context.ToDocument<Forum>(forum101);
```

Especificación de parámetros opcionales para DynamoDBContext

Cuando se utiliza el modelo de persistencia de objetos, es posible especificar los siguientes parámetros opcionales para la clase `DynamoDBContext`.

- **`ConsistentRead`**: cuando se recuperan datos utilizando las operaciones `Load`, `Query` o `Scan`, es posible agregar este parámetro si se desea para solicitar los valores más recientes de los datos.
- **`IgnoreNullValues`**: este parámetro informa a `DynamoDBContext` de que debe pasar por alto los valores null de los atributos durante una operación `Save`. Si este parámetro es false (o, si no se ha establecido), entonces un valor null se interpretará como una instrucción de eliminar el atributo de que se trate.
- **`SkipVersionCheck`**: este parámetro informa a `DynamoDBContext` de que no debe comparar las versiones al guardar o eliminar un elemento. Para obtener más información sobre el control de versiones, consulte Bloqueo positivo mediante DynamoDB y el modelo de persistencia de objetos de AWS SDK para .NET.
- **`TableNamePrefix`**— antepone una cadena determinada a los nombres de todas las tablas. Si este parámetro es null (o si no se ha establecido), no se utilizará ningún prefijo.
- `DynamoDBEntryConversion`: especifica el esquema de conversión que utiliza el cliente. Puede establecer este parámetro en la versión V1 o V2. V1 es la versión predeterminada.

  En función de la versión que establezca, el comportamiento de este parámetro cambia. Por ejemplo:

  - En V1, el tipo de datos `bool` se convierte al tipo numérico `N`, donde 0 representa falso y 1 representa verdadero. En V2, `bool` se convierte en `BOOL`.
  - En V2, las listas y las matrices no se agrupan con los HashSets. Las listas y las matrices de números, tipos basados en cadenas y tipos basados en binarios se convierten al tipo `L` (Lista), que se puede enviar vacío para actualizar una lista. A diferencia de V1, en la que no se envía una lista vacía por cable.

En V1, los tipos de recopilaciones, como List, HashSet y matrices, se tratan de la misma manera. La lista, el HashSet y la matriz de números se convierten al tipo NS (conjunto de números).

El siguiente ejemplo establece la versión del esquema de conversión en V2, lo que cambia el comportamiento de conversión entre los tipos .NET y los tipos de datos de DynamoDB.

```
var config = new DynamoDBContextConfig
{
    Conversion = DynamoDBEntryConversion.V2
};
var contextV2 = new DynamoDBContext(client, config);
```

En los siguientes fragmentos de código C# se crea un nuevo `DynamoDBContext` especificando dos de los parámetros opcionales anteriores, `ConsistentRead` y `SkipVersionCheck`.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context =
       new DynamoDBContext(client, new DynamoDBContextConfig { ConsistentRead = true,
 SkipVersionCheck = true});
```

`DynamoDBContext` incluye estos parámetros opcionales con cada solicitud que se envía utilizando este contexto.

En lugar de establecer estos parámetros en el nivel de `DynamoDBContext`, puede especificarlos para las operaciones individuales que ejecute utilizando `DynamoDBContext`, como se muestra en el siguiente ejemplo de código C#. En el ejemplo se carga un elemento de libro concreto. El método `Load` de `DynamoDBContext` especifica los parámetros opcionales `ConsistentRead` y `SkipVersionCheck`.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
...
DynamoDBContext context = new DynamoDBContext(client);
```

```
Book bookItem = context.Load<Book>(productId,new DynamoDBContextConfig{ ConsistentRead
 = true, SkipVersionCheck = true });
```

En este caso, `DynamoDBContext` incluye estos parámetros solo cuando se envía la solicitud `Get`.

Bloqueo positivo mediante DynamoDB y el modelo de persistencia de objetos de AWS SDK para .NET

La compatibilidad del modelo de persistencia de objetos con el bloqueo optimista garantiza que la versión del elemento en la aplicación sea la misma que en el lado del servidor antes de actualizar o eliminar el elemento. Supongamos que recupera un elemento para actualizarlo. Sin embargo, antes de que se devuelvan las actualizaciones, otra aplicación actualiza el mismo elemento. Ahora, su aplicación tiene una copia anticuada del elemento. Sin el bloqueo optimista, cualquier actualización que lleve a cabo sobrescribirá la actualización efectuada por la otra aplicación.

La característica de bloqueo optimista del modelo de persistencia de objetos proporciona la etiqueta `DynamoDBVersion`, que se puede usar para habilitar el bloqueo optimista. Para utilizar esta característica, se agrega una propiedad a la clase para almacenar el número de versión. Agregue el atributo `DynamoDBVersion` a la propiedad. La primera vez que se guarda el objeto, `DynamoDBContext` asigna un número de versión e incrementa este valor cada vez que se actualiza el elemento.

Su solicitud de actualización o eliminación solamente se llevará a cabo si la versión del objeto en el lado del cliente coincide con el número de versión del elemento correspondiente en el lado del servidor. Si su aplicación contiene una copia anticuada, debe obtener la versión más reciente del servidor para poder actualizar o eliminar el elemento en cuestión.

En el siguiente ejemplo de código C# se define una clase `Book` con atributos de persistencia de objetos que la mapean a la tabla `ProductCatalog`. La propiedad `VersionNumber` de la clase asociada con el atributo `DynamoDBVersion` almacena el valor del número de versión.

Example

```
[DynamoDBTable("ProductCatalog")]
  public class Book
  {
    [DynamoDBHashKey]   //Partition key
    public int Id { get; set; }
    [DynamoDBProperty]
    public string Title { get; set; }
    [DynamoDBProperty]
```

```
    public string ISBN { get; set; }
    [DynamoDBProperty("Authors")]
    public List<string> BookAuthors { get; set; }
    [DynamoDBVersion]
    public int? VersionNumber { get; set; }
  }
```

> **Note**
>
> Puede aplicar el atributo `DynamoDBVersion` solamente a un tipo numérico primitivo que pueda contener valores null (por ejemplo, `int?`).

El bloqueo optimista afecta a las operaciones de `DynamoDBContext` como se indica a continuación:

- Para un elemento nuevo, `DynamoDBContext` asigna el número de versión inicial 0. Si recupera un elemento, actualiza una o varias de sus propiedades e intenta guardar los cambios, la operación de almacenamiento solamente se lleva a cabo si el número de versión del lado del cliente coincide con el número de versión del lado del servidor. `DynamoDBContext` aumenta el número de versión. No es necesario establecer el número de versión.
- El método `Delete` proporciona sobrecargas que pueden tomar un valor de clave principal o un objeto como parámetro, como se muestra en el siguiente ejemplo de código C#.

  Example

  ```
  DynamoDBContext context = new DynamoDBContext(client);
  ...
  // Load a book.
  Book book = context.Load<ProductCatalog>(111);
  // Do other operations.
  // Delete 1 - Pass in the book object.
  context.Delete<ProductCatalog>(book);

  // Delete 2 - Pass in the Id (primary key)
  context.Delete<ProductCatalog>(222);
  ```

  Si proporciona un objeto como parámetro, la eliminación solamente se llevará a cabo si la versión del objeto coincide con el número de versión del elemento correspondiente en el lado del servidor. Sin embargo, si proporciona un valor de clave principal como parámetro, `DynamoDBContext` no detectará los números de versión y eliminará el elemento sin comprobar la versión.

Tenga en cuenta que la implementación interna del bloqueo optimista en el código del modelo de persistencia de objetos utiliza las acciones de actualización condicional y eliminación condicional del API de DynamoDB.

Deshabilitación del bloqueo positivo

Para deshabilitar el bloqueo optimista, se usa la propiedad de configuración `SkipVersionCheck`. Puede establecer esta propiedad al crear `DynamoDBContext`. En este caso, el bloqueo optimista está deshabilitado para todas las solicitudes que se realicen utilizando el contexto. Para obtener más información, consulte Especificación de parámetros opcionales para DynamoDBContext .

En lugar de establecer la propiedad para todo el contexto, puede deshabilitar el bloqueo optimista para una operación específica, tal y como se muestra en el siguiente ejemplo de código C#. En el ejemplo de código se utiliza el contexto para eliminar un elemento de libro. El método `Delete` establece la propiedad `SkipVersionCheck` opcional en true, con lo que deshabilita la comprobación de versiones.

Example

```
DynamoDBContext context = new DynamoDBContext(client);
// Load a book.
Book book = context.Load<ProductCatalog>(111);
...
// Delete the book.
context.Delete<Book>(book, new DynamoDBContextConfig { SkipVersionCheck = true });
```

Mapeo de datos arbitrarios con DynamoDB mediante el modelo de persistencia de objetos de AWS SDK para .NET

Además de los tipos de .NET admitidos (consulte Tipos de datos compatibles), puede utilizar tipos de la aplicación para los cuales no exista un mapeo directo a los tipos de Amazon DynamoDB. El modelo de persistencia de objetos es compatible con el almacenamiento de datos de tipos arbitrarios, siempre y cuando se proporcione el convertidor requerido para convertir los datos del tipo arbitrario al tipo de DynamoDB y viceversa. El código del convertidor transforma los datos tanto al guardar como al cargar los objetos.

Puede crear cualquier tipo en el lado del cliente. Sin embargo, los datos almacenados en las tablas serán de uno de los tipos de DynamoDB. Además, durante las consultas y los exámenes, las comparaciones se harán respecto a los datos almacenados en DynamoDB.

En el siguiente ejemplo de código C# se define una clase `Book` con las propiedades `Id`, `Title`, `ISBN` y `Dimension`. La propiedad `Dimension` es del tipo `DimensionType`, que describe las propiedades `Height`, `Width` y `Thickness`. En el ejemplo de código se proporcionan los métodos de convertidor `ToEntry` y `FromEntry` para convertir los datos entre el tipo `DimensionType` y el tipo String de DynamoDB. Por ejemplo, al,x guardar una instancia `Book`, el convertidor crea una cadena `Dimension` de libro tal como “8,5x11x0,05”. Cuando recupera un libro, convierte la cadena a una instancia `DimensionType`.

En el ejemplo se mapea el tipo `Book` a la tabla `ProductCatalog`. Guarda un ejemplo de instancia de `Book`, la recupera, actualiza sus dimensiones y vuelve a guardar `Book` una vez actualizado.

Para obtener instrucciones paso a paso para probar el siguiente ejemplo, consulte Ejemplos de código .NET.

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class HighLevelMappingArbitraryData
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                DynamoDBContext context = new DynamoDBContext(client);

                // 1. Create a book.
                DimensionType myBookDimensions = new DimensionType()
                {
                    Length = 8M,
                    Height = 11M,
                    Thickness = 0.5M
```

```
                };

                Book myBook = new Book
                {
                    Id = 501,
                    Title = "AWS SDK for .NET Object Persistence Model Handling
Arbitrary Data",
                    ISBN = "999-9999999999",
                    BookAuthors = new List<string> { "Author 1", "Author 2" },
                    Dimensions = myBookDimensions
                };

                context.Save(myBook);

                // 2. Retrieve the book.
                Book bookRetrieved = context.Load<Book>(501);

                // 3. Update property (book dimensions).
                bookRetrieved.Dimensions.Height += 1;
                bookRetrieved.Dimensions.Length += 1;
                bookRetrieved.Dimensions.Thickness += 0.2M;
                // Update the book.
                context.Save(bookRetrieved);

                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }
    }
    [DynamoDBTable("ProductCatalog")]
    public class Book
    {
        [DynamoDBHashKey] //Partition key
        public int Id
        {
            get; set;
        }
        [DynamoDBProperty]
        public string Title
        {
            get; set;
```

```
        }
        [DynamoDBProperty]
        public string ISBN
        {
            get; set;
        }
        // Multi-valued (set type) attribute.
        [DynamoDBProperty("Authors")]
        public List<string> BookAuthors
        {
            get; set;
        }
        // Arbitrary type, with a converter to map it to DynamoDB type.
        [DynamoDBProperty(typeof(DimensionTypeConverter))]
        public DimensionType Dimensions
        {
            get; set;
        }
    }

    public class DimensionType
    {
        public decimal Length
        {
            get; set;
        }
        public decimal Height
        {
            get; set;
        }
        public decimal Thickness
        {
            get; set;
        }
    }

    // Converts the complex type DimensionType to string and vice-versa.
    public class DimensionTypeConverter : IPropertyConverter
    {
        public DynamoDBEntry ToEntry(object value)
        {
            DimensionType bookDimensions = value as DimensionType;
            if (bookDimensions == null) throw new ArgumentOutOfRangeException();
```

```
            string data = string.Format("{1}{0}{2}{0}{3}", " x ",
                            bookDimensions.Length, bookDimensions.Height,
 bookDimensions.Thickness);

            DynamoDBEntry entry = new Primitive
            {
                Value = data
            };
            return entry;
        }

        public object FromEntry(DynamoDBEntry entry)
        {
            Primitive primitive = entry as Primitive;
            if (primitive == null || !(primitive.Value is String) ||
 string.IsNullOrEmpty((string)primitive.Value))
                throw new ArgumentOutOfRangeException();

            string[] data = ((string)(primitive.Value)).Split(new string[] { " x " },
 StringSplitOptions.None);
            if (data.Length != 3) throw new ArgumentOutOfRangeException();

            DimensionType complexData = new DimensionType
            {
                Length = Convert.ToDecimal(data[0]),
                Height = Convert.ToDecimal(data[1]),
                Thickness = Convert.ToDecimal(data[2])
            };
            return complexData;
        }
    }
}
```

Cómo ejecutar los ejemplos de código de esta guía para desarrolladores

Los SDK de AWS proporcionan un amplio soporte para Amazon DynamoDB en los siguientes lenguajes:

- Java
- JavaScript en el navegador
- .NET
- Node.js

- PHP
- Python
- Ruby
- C++
- Go
- Android
- iOS

Los ejemplos de código de esta guía para desarrolladores ofrecen una cobertura más exhaustiva de las operaciones de DynamoDB en los siguientes lenguajes de programación:

- Ejemplos de código Java
- Ejemplos de código .NET

Antes de comenzar con este ejercicio, es preciso crear una cuenta de AWS, obtener la clave de acceso y la clave secreta, y configurar la AWS Command Line Interface (AWS CLI) en el ordenador. Para obtener más información, consulte Configuración de DynamoDB (servicio web) .

> **Note**
>
> Si va a utilizar la versión descargable de DynamoDB, debe utilizar la AWS CLI para crear las tablas y los ejemplos de datos. Asimismo, debe especificar el parámetro `--endpoint-url` en cada comando de la AWS CLI. Para obtener más información, consulte Configuración del punto de conexión local .

Creación de tablas y carga de datos para ejemplos de código en DynamoDB

Consulte a continuación los conceptos básicos sobre la creación de tablas en DynamoDB, la carga de un conjunto de datos de ejemplo, la consulta de los datos y su actualización.

- Paso 1: crear una tabla en DynamoDB
- Paso 2: escribir datos en una tabla de DynamoDB
- Paso 3: leer datos de una tabla de DynamoDB
- Paso 4: actualizar los datos de una tabla de DynamoDB

Ejemplos de código Java

Temas

- Java: configuración de las credenciales de AWS
- Java: configuración de la región y del punto de conexión de AWS

Esta Guía para desarrolladores contiene fragmentos de código Java y programas listos para ejecutarlos. Encontrará estos ejemplos de código en las secciones siguientes:

- Uso de elementos y atributos en DynamoDB
- Uso de tablas y datos en DynamoDB
- Consulta de tablas en DynamoDB
- Análisis de tablas en DynamoDB
- Mejora del acceso con índices secundarios en DynamoDB
- Java 1.x: DynamoDBMapper
- Captura de datos de cambios para DynamoDB Streams

Puede comenzar rápidamente a trabajar con Eclipse y AWS Toolkit for Eclipse. Además de un IDE completo, obtendrá AWS SDK para Java con actualizaciones automáticas y plantillas preconfiguradas para crear aplicaciones de AWS.

Para ejecutar los ejemplos de código Java (con Eclipse)

1. Descargue e instale el IDE de Eclipse.
2. Descargue e instale AWS Toolkit for Eclipse.
3. Inicie Eclipse y, en el menú Eclipse, elija File (Archivo), New (Nuevo) y después Other (Otro).
4. En Seleccionar un asistente, elija AWS, Proyecto de Java de AWS y, después, Siguiente.
5. En Crear una instancia de Java de AWS, haga lo siguiente:

    a. En Nombre del proyecto, introduzca un nombre para el proyecto.

    b. En Select Account (Seleccionar cuenta), elija su perfil de credenciales en la lista.

    Si es la primera vez que utiliza AWS Toolkit for Eclipse, elija Configurar cuentas de AWS para configurar las credenciales de AWS.

6. Elija Finish (Finalizar) para crear el proyecto.
7. En el menú Eclipse, elija File (Archivo), New (Nuevo) y después Class (Clase).
8. En Java Class (Clase Java), introduzca un nombre para la clase en Name (Nombre) (use el mismo nombre que en el ejemplo de código que desea ejecutar) y, a continuación, elija Finish (Finalizar) para crear la clase.
9. Copie el ejemplo de código de la página de documentación en el editor de Eclipse.
10. Para ejecutar el código, elija Run (Ejecutar) en el menú Eclipse.

El SDK para Java proporciona clientes seguros para subprocesos con el fin de trabajar con DynamoDB. Como práctica recomendada, sus aplicaciones deben crear un cliente y reutilizar el cliente entre subprocesos.

Para obtener más información, consulte AWS SDK para Java.

> **Note**
>
> Los ejemplos de código de esta guía se han diseñado para utilizarlos con la última versión del AWS SDK para Java.
> Si utiliza AWS Toolkit for Eclipse, puede configurar las actualizaciones automáticas para el SDK para Java. Para hacer esto en Eclipse, vaya a Preferences (Preferencias) y elija AWS Toolkit, AWS SDK para Java, Download new SDKs automatically (Descargar nuevos SDK automáticamente).

Java: configuración de las credenciales de AWS

El SDK para Java requiere que proporcione las credenciales de AWS a su aplicación en tiempo de ejecución. En los ejemplos de código de esta guía se supone que se usa un archivo de credenciales de AWS, tal como se describe en Configuración de las credenciales de AWS en la Guía para desarrolladores de AWS SDK para Java.

A continuación se muestra un ejemplo de archivo de credenciales de AWS denominado `~/.aws/credentials`, donde el carácter de tilde (~) representa su directorio de inicio.

```
[default]
aws_access_key_id = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```

Java: configuración de la región y del punto de conexión de AWS

De forma predeterminada, los ejemplos de código acceden a DynamoDB en la región EE. UU. Oeste (Oregón). Puede cambiar la región modificando las propiedades de `AmazonDynamoDB`.

En el siguiente ejemplo de código se crea una nueva instancia de `AmazonDynamoDB`.

```
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import com.amazonaws.regions.Regions;
...
// This client will default to US West (Oregon)
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
.withRegion(Regions.US_WEST_2)
.build();
```

Puede usar el método `withRegion` para ejecutar el código en DynamoDB en cualquier región donde se encuentre disponible. Para obtener una lista completa, consulte Regiones de AWS y puntos de conexión en la Referencia general de Amazon Web Services.

Si desea ejecutar los ejemplos de código en DynamoDB localmente en su ordenador, configure el punto de enlace como se indica a continuación.

SDK de AWS V1

```
AmazonDynamoDB client =
 AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(
new AwsClientBuilder.EndpointConfiguration("http://localhost:8000", "us-west-2"))
.build();
```

SDK de AWS V2

```
DynamoDbClient client = DynamoDbClient.builder()
    .endpointOverride(URI.create("http://localhost:8000"))
    // The region is meaningless for local DynamoDb but required for client builder
 validation
    .region(Region.US_EAST_1)
    .credentialsProvider(StaticCredentialsProvider.create(
    AwsBasicCredentials.create("dummy-key", "dummy-secret")))
    .build();
```

Ejemplos de código .NET

Temas

- .NET: configuración de las credenciales de AWS
- .NET: configuración de la región y del punto de conexión de AWS

Esta guía contiene fragmentos de código .NET y programas listos para ejecutarlos. Encontrará estos ejemplos de código en las secciones siguientes:

- Uso de elementos y atributos en DynamoDB
- Uso de tablas y datos en DynamoDB
- Consulta de tablas en DynamoDB
- Análisis de tablas en DynamoDB
- Mejora del acceso con índices secundarios en DynamoDB
- Uso del modelo de documento de .NET en DynamoDB
- Uso del modelo de persistencia de objetos de .NET y DynamoDB
- Captura de datos de cambios para DynamoDB Streams

Puede comenzar rápidamente a trabajar utilizando AWS SDK para .NET con Toolkit for Visual Studio.

Para ejecutar los ejemplos de código .NET (mediante Visual Studio)

1. Descargue e instale Microsoft Visual Studio.
2. Descargar e instalar Toolkit for Visual Studio.
3. Inicie Visual Studio. Elija File (Archivo), New (Nuevo), Project (Proyecto).
4. En Nuevo proyecto, elija Proyecto vacío de AWS y, después, elija Aceptar.
5. En Credenciales de acceso de AWS, elija Usar perfil existente, elija el perfil de sus credenciales de la lista y, a continuación, elija Aceptar.

   Si es la primera vez que utiliza Toolkit for Visual Studio, elija Use a new profile (Usar un nuevo perfil) para configurar las credenciales de AWS.

6. En el proyecto de Visual Studio, elija la pestaña correspondiente al código fuente del programa (`Program.cs`). Copie el ejemplo de código de la página de documentación en el editor de Visual Studio sustituyendo cualquier otro código que aparezca en el editor.
7. Si aparece algún mensaje de error como The type or namespace name...could not be found (El tipo o el nombre de espacio de nombres...no se ha podido encontrar), debe instalar el ensamblado del SDK de AWS para DynamoDB como se indica a continuación:
   a. En el Explorador de soluciones, abra el menú contextual (haga clic con el botón derecho) del proyecto y elija Administrar paquetes NuGet.
   b. En Administrador de paquetes NuGet, elija Examinar.
   c. En el cuadro de búsqueda, introduzca **`AWSSDK.DynamoDBv2`** y espere a que se lleve a cabo la búsqueda.
   d. Elija AWSSDK.DynamoDBv2 y después elija Install (Instalar).
   e. Cuando se haya completado la instalación, elija la pestaña Program.cs para volver al programa.
8. Para ejecutar el código, elija Iniciar en la barra de herramientas de Visual Studio.

SDK para .NET proporciona clientes seguros para subprocesos con el fin de trabajar con DynamoDB. Como práctica recomendada, sus aplicaciones deben crear un cliente y reutilizar el cliente entre subprocesos.

Para obtener más información, consulte el SDK para .NET de AWS.

> **Note**
>
> Los ejemplos de código de esta guía se han diseñado para utilizarlos con la última versión del AWS SDK para .NET.

.NET: configuración de las credenciales de AWS

SDK para .NET requiere que proporcione las credenciales de AWS a su aplicación en tiempo de ejecución. En los ejemplos de código de esta guía se supone que se usa el SDK Store para administrar el archivo de credenciales de AWS, tal como se describe en Uso del SDK Store en la Guía para desarrolladores de AWS SDK para .NET.

Toolkit for Visual Studio admite varios conjuntos de credenciales de cualquier cantidad de cuentas. Cada conjunto se conoce como perfil. Visual Studio agrega entradas al archivo `App.config` del proyecto, para que la aplicación encuentre las credenciales de AWS en tiempo de ejecución.

En el siguiente ejemplo se muestra el archivo `App.config` predeterminado que se genera al crear un nuevo proyecto con Toolkit for Visual Studio.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
    <add key="AWSProfileName" value="default"/>
    <add key="AWSRegion" value="us-west-2" />
 </appSettings>
</configuration>
```

En tiempo de ejecución, el programa usa el conjunto `default` de credenciales de AWS, según lo especificado en la entrada `AWSProfileName`. Las credenciales de AWS en sí se conservan en SDK Store, cifradas. Toolkit for Visual Studio proporciona una interfaz gráfica de usuario para administrar las credenciales, todo ello desde Visual Studio. Para obtener más información, consulte Especificación de credenciales en la Guía del usuario de AWS Toolkit for Visual Studio.

> **Note**
>
> De forma predeterminada, los ejemplos de código acceden a DynamoDB en la región EE. UU. Oeste (Oregón). Puede cambiar la región modificando la entrada `AWSRegion` en el archivo App.config. Puede establecer `AWSRegion` en cualquier región donde DynamoDB esté disponible. Para obtener una lista completa, consulte Regiones de AWS y puntos de conexión en la Referencia general de Amazon Web Services.

.NET: configuración de la región y del punto de conexión de AWS

De forma predeterminada, los ejemplos de código acceden a DynamoDB en la región EE. UU. Oeste (Oregón). Puede cambiar la región modificando la entrada `AWSRegion` en el archivo `App.config`. O bien puede cambiar la región modificando las propiedades de `AmazonDynamoDBClient`.

En el siguiente ejemplo de código se crea una nueva instancia de `AmazonDynamoDBClient`. El cliente se modifica de tal forma que el código se ejecute en DynamoDB en otra región.

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
```

```
// This client will access the US East 1 region.
clientConfig.RegionEndpoint = RegionEndpoint.USEast1;
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

Para obtener una lista completa de las regiones, consulte Regiones de AWS y puntos de conexión en la Referencia general de Amazon Web Services.

Si desea ejecutar los ejemplos de código en DynamoDB localmente en su ordenador, configure el punto de enlace como se indica a continuación.

```
AmazonDynamoDBConfig clientConfig = new AmazonDynamoDBConfig();
// Set the endpoint URL
clientConfig.ServiceURL = "http://localhost:8000";
AmazonDynamoDBClient client = new AmazonDynamoDBClient(clientConfig);
```

API de bajo nivel de DynamoDB

La API de bajo nivel de Amazon DynamoDB es la interfaz de nivel de protocolo de DynamoDB. En este nivel, cada solicitud HTTP(S) debe tener el formato correcto y llevar una firma digital válida.

Los SDK de AWS construyen las solicitudes a la API de bajo nivel de DynamoDB automáticamente y procesan las respuestas de DynamoDB. Esto le permite centrarse en la lógica de la aplicación, en lugar de en los detalles de bajo nivel. Sin embargo, le resultará útil conocer algunos conceptos básicos del funcionamiento de la API de bajo nivel de DynamoDB.

Para obtener más información sobre la API de DynamoDB de bajo nivel, consulteReferencia de la API de Amazon DynamoDB.

> **Note**
>
> DynamoDB Streams tiene su propio API de bajo nivel, que es independiente de la de DynamoDB y totalmente compatible con los SDK de AWS.
> Para obtener más información, consulte Captura de datos de cambios para DynamoDB Streams. Para la API de DynamoDB Streams de bajo nivel, consulte la. Referencia de la API de Amazon DynamoDB Streams.

La API de bajo nivel de DynamoDB utiliza la notación de objetos de JavaScript (JSON, JavaScript Object Notation) como formato de protocolo de conexión. JSON presenta los datos de forma

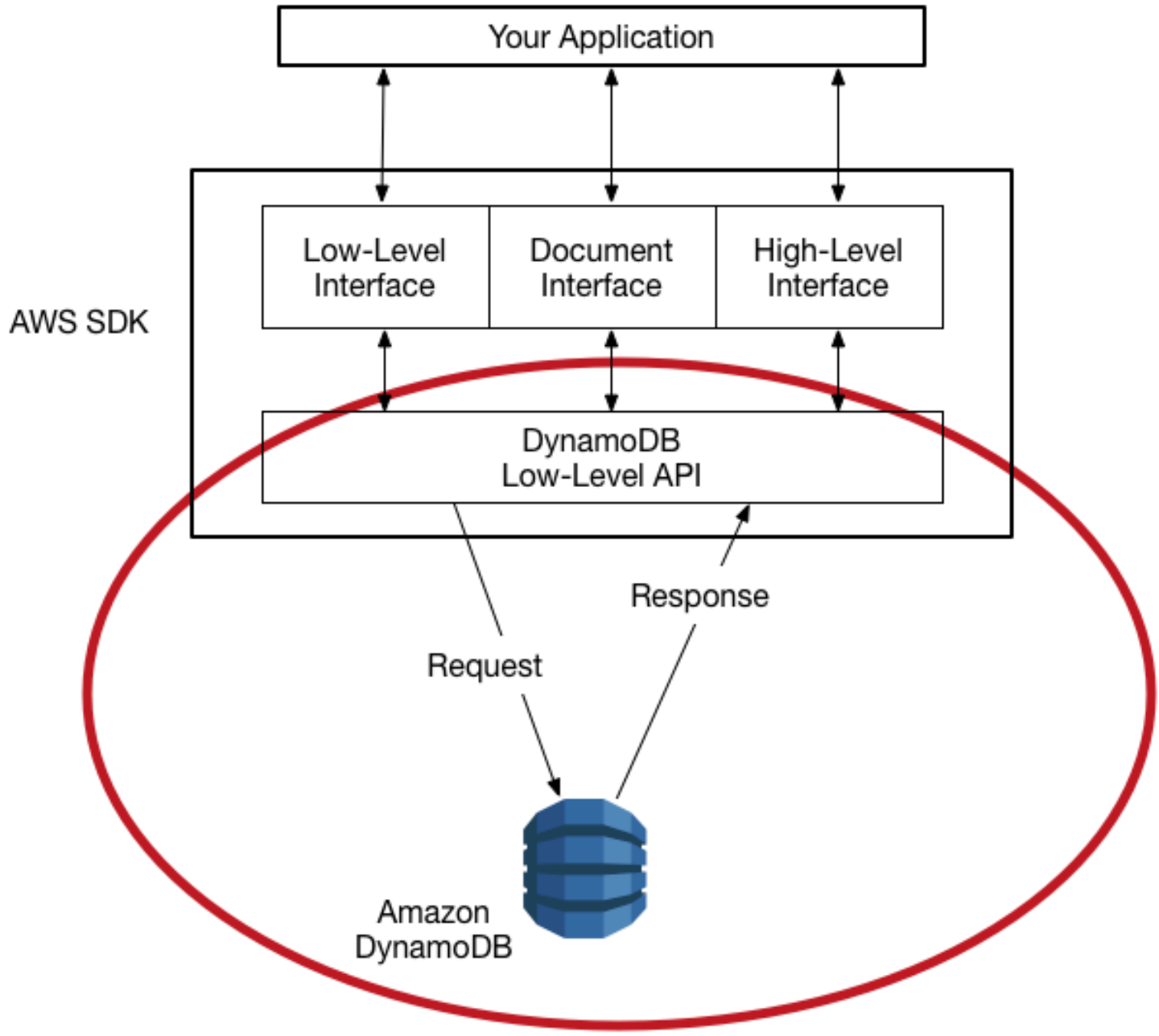
jerárquica para transmitir simultáneamente sus valores y su estructura. Los pares de nombre-valor se definen con el formato `name:value`. La jerarquía de datos se define mediante llaves anidadas de pares de nombre-valor.

DynamoDB usa JSON como protocolo de transporte únicamente, no como formato de almacenamiento. Los SDK de AWS usan JSON para enviar datos a DynamoDB y DynamoDB responde con JSON. DynamoDB no almacena datos de forma persistente en formato JSON.

> **Note**
>
> Para obtener más información acerca de JSON, consulte Introducing JSON en el sitio web `JSON.org`.

Temas

- Formato de solicitudes
- Formato de las respuestas
- Descriptores de tipos de datos
- Datos numéricos
- Datos binarios



Formato de solicitudes

La API de bajo nivel de DynamoDB acepta solicitudes `POST` de HTTP(S) como información de entrada. Los SDK de AWS construyen estas solicitudes automáticamente.

Supongamos que dispone de una tabla denominada `Pets`, con un esquema de claves que consta de `AnimalType` (clave de partición) y `Name` (clave de ordenación). Ambos atributos son de tipo `string`. Para recuperar un elemento de `Pets`, el SDK de AWS construye la siguiente solicitud.

```
POST / HTTP/1.1
```

```
Host: dynamodb.<region>.<domain>;
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
X-Amz-Date: <Date>
X-Amz-Target: DynamoDB_20120810.GetItem

{
    "TableName": "Pets",
    "Key": {
        "AnimalType": {"S": "Dog"},
        "Name": {"S": "Fido"}
    }
}
```

Tenga en cuenta lo siguiente en relación con esta solicitud:

- El encabezado `Authorization` contiene la información necesaria para que DynamoDB autentique la solicitud. Para obtener más información, consulte Firma de solicitudes de API AWS y Proceso de firma de Signature Version 4 en la Referencia general de Amazon Web Services.
- El encabezado `X-Amz-Target` contiene el nombre de una operación de DynamoDB, `GetItem`. Además, va acompañada de la versión del API de bajo nivel, en este caso, `20120810`.
- La carga (cuerpo) de la solicitud contiene los parámetros de la operación, en formato JSON. En la operación `GetItem`, los parámetros son `TableName` y `Key`.

Formato de las respuestas

Una vez que recibe la solicitud, DynamoDB la procesa y devuelve una respuesta. Para la solicitud mostrada anteriormente, la carga de la respuesta HTTP(S) contiene los resultados de la operación, como se muestra en el siguiente ejemplo.

```
HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

```
{
    "Item": {
        "Age": {"N": "8"},
        "Colors": {
            "L": [
                {"S": "White"},
                {"S": "Brown"},
                {"S": "Black"}
            ]
        },
        "Name": {"S": "Fido"},
        "Vaccinations": {
            "M": {
                "Rabies": {
                    "L": [
                        {"S": "2009-03-17"},
                        {"S": "2011-09-21"},
                        {"S": "2014-07-08"}
                    ]
                },
                "Distemper": {"S": "2015-10-13"}
            }
        },
        "Breed": {"S": "Beagle"},
        "AnimalType": {"S": "Dog"}
    }
}
```

En este momento, el SDK de AWS devuelve los datos de respuesta a la aplicación para continuar procesándolos.

> **Note**
>
> Si DynamoDB no puede procesar una solicitud, devuelve un código de error HTTP y un mensaje. El SDK de AWS los propaga a su aplicación, en forma de excepciones. Para obtener más información, consulte Control de errores con DynamoDB.

Descriptores de tipos de datos

El protocolo de la API de bajo nivel de DynamoDB requiere que cada atributo vaya acompañado de un descriptor del tipo de datos. Los descriptores de tipos de datos son tokens que indican a DynamoDB cómo interpretar cada atributo.

En las secciones Formato de solicitudes y Formato de las respuestas encontrará ejemplos de cómo se usan los descriptores de tipos de datos. En la solicitud `GetItem` se especifica `S` para los atributos del esquema de claves de `Pets` (`AnimalType` y `Name`), que son de tipo `string`. La respuesta `GetItem` contiene un elemento Pets con atributos de tipo `string` (`S`), `number` (`N`), `map` (`M`) y `list` (`L`).

A continuación se muestra una lista completa de descriptores de tipos de datos de DynamoDB:

- **`S`**: String
- **`N`**: Number
- **`B`**: Binary
- **`BOOL`**: Boolean
- **`NULL`**: Null
- **`M`**: Map
- **`L`**: List
- **`SS`**: String Set
- **`NS`**: Number Set
- **`BS`**: Binary Set

> **Note**
>
> Para obtener descripciones detalladas de los tipos de datos de DynamoDB, consulte Tipos de datos.

Datos numéricos

Los distintos lenguajes de programación ofrecen diferentes niveles de compatibilidad con JSON. En algunos casos, es posible que prefiera usar una biblioteca de terceros para validar y analizar los documentos JSON.

Algunas bibliotecas de terceros se basan en el tipo Number de JSON y proporcionan sus propios tipos, tales como `int`, `long` o `double`. Sin embargo, el tipo de datos Number nativo de DynamoDB no se mapea exactamente a estos otros tipos de datos, por lo que estas diferencias entre los tipos pueden provocar conflictos. Además, muchas bibliotecas JSON no controlan los valores numéricos con una precisión fija y deducen automáticamente que el tipo de datos de las secuencias que contienen una coma decimal es double.

Para resolver estos problemas, DynamoDB proporciona un único tipo numérico sin pérdida de datos. Para evitar conversiones implícitas no deseadas a un valor de tipo double, DynamoDB utiliza cadenas para efectuar la transferencia de datos de valores numéricos. Este enfoque proporciona flexibilidad para actualizar los valores de los atributos y, al mismo tiempo, mantener una semántica de ordenación adecuada; por ejemplo, colocar los valores "01", "2" y "03" en la secuencia correcta.

Si la precisión del número es importante para la aplicación, debe convertir los valores numéricos en cadenas antes de pasárselos a DynamoDB.

Datos binarios

DynamoDB es compatible con los atributos binarios. Sin embargo, JSON no presenta compatibilidad nativa con la codificación de datos binarios. Para enviar datos binarios en una solicitud, es preciso codificarlos en formato base64. Al recibir la solicitud, DynamoDB decodifica los datos base64 para que vuelvan a ser binarios.

El esquema de codificación base64 que se utiliza en DynamoDB se describe en RFC 4648 en el sitio web de IETF (Internet Engineering Task Force).

Programación de Amazon DynamoDB con Python y Boto3

Esta guía ofrece orientación a los programadores que desean utilizar Amazon DynamoDB con Python. Obtenga información sobre las distintas capas de abstracción, la configuración de la administración, la gestión de errores, el control de las políticas de reintentos, la gestión de keep-alive y mucho más.

Temas

- Acerca de Boto
- Uso de la documentación de Boto
- Comprensión de las capas de abstracción de clientes y recursos

- Uso del recurso de tabla batch_writer
- Ejemplos de código adicionales que exploran las capas de cliente y de recursos
- Comprensión sobre la interacción de los objetos Client y Resource con las sesiones y los subprocesos
- Personalización del objeto Config
- Gestión de errores
- Registro
- Enlaces de eventos
- La paginación y el paginador
- Esperadores

Acerca de Boto

Puede acceder a DynamoDB desde Python desde el SDK oficial de AWS para Python, conocido comúnmente como Boto3. El nombre Boto proviene de un delfín de agua dulce originario del río Amazonas. La biblioteca Boto3 es la tercera versión principal de la biblioteca publicada por primera vez en 2015. La biblioteca Boto3 es bastante grande, ya que es compatible con todos los servicios de AWS, no solo con DynamoDB. Esta guía trata únicamente las partes de Boto3 relevantes para DynamoDB.

AWS actualiza y publica Boto como proyecto de código abierto alojado en GitHub, que está dividido en dos paquetes: Botocore y Boto3.

- Botocore proporciona la funcionalidad de bajo nivel. En Botocore encontrará las clases de cliente, sesión, credenciales, configuración y excepción.
- Boto3 se basa en Botocore y ofrece una interfaz más cercana a Python y de nivel superior. En concreto, expone una tabla de DynamoDB como recurso y ofrece una interfaz más sencilla y elegante en comparación con la interfaz de cliente orientada a servicios de nivel inferior.

Dado que estos proyectos están alojados en GitHub, puede ver el código fuente, realizar un seguimiento de los problemas pendientes o enviar sus propios problemas.

Uso de la documentación de Boto

Comience a utilizar la documentación de Boto con los siguientes recursos:

- Comience con la sección Quickstart, que proporciona un punto de partida sólido para instalar el paquete. Navegue hasta allí para obtener instrucciones sobre cómo instalar Boto3 si aún no lo ha instalado (Boto3 suele estar disponible automáticamente en los servicios de AWS tales como AWS Lambda).
- Después, céntrese en la documentación de la guía de DynamoDB donde se explica cómo realizar las actividades básicas de DynamoDB: crear y eliminar una tabla, manipular elementos, ejecutar operaciones por lotes, ejecutar una consulta y realizar un análisis. Los ejemplos utilizan la interfaz de recursos. Cuando vea `boto3.resource('dynamodb')`, significa que está utilizando la interfaz de recursos de nivel superior.
- Tras leer la guía, puede consultar la referencia de DynamoDB. Esta página de aterrizaje proporciona una lista exhaustiva de las clases y los métodos disponibles. En la parte superior, verá la clase `DynamoDB.Client` que proporciona un acceso de bajo nivel a todas las operaciones del plano de control y del plano de datos. En la parte inferior, verá la clase `DynamoDB.ServiceResource`. Esta es la interfaz de Python de nivel superior. Con ella, puede crear una tabla, realizar operaciones por lotes entre tablas u obtener una instancia `DynamoDB.ServiceResource.Table` para acciones específicas de una tabla.

Comprensión de las capas de abstracción de clientes y recursos

Las dos interfaces con las que trabajará son la interfaz de cliente y la interfaz de recursos.

- La interfaz de cliente de bajo nivel proporciona una asignación uno a uno con la API del servicio subyacente. Todas las API que ofrece DynamoDB están disponibles a través del cliente. Esto significa que la interfaz de cliente puede proporcionar toda la funcionalidad, pero suele ser más detallada y su uso es más complejo.
- La interfaz de recursos de nivel superior no proporciona una asignación uno a uno de la API del servicio subyacente. Sin embargo, proporciona métodos que facilitan el acceso al servicio, como `batch_writer`.

A continuación, se muestra un ejemplo de cómo insertar un elemento mediante la interfaz de cliente. Observe cómo todos los valores se transfieren como un mapa: la clave indica el tipo («S» para cadena, «N» para número) y el valor es de cadena. Este es el formato JSON de DynamoDB.

```
import boto3

dynamodb = boto3.client('dynamodb')
```

```
dynamodb.put_item(
    TableName='YourTableName',
    Item={
        'pk': {'S': 'id#1'},
        'sk': {'S': 'cart#123'},
        'name': {'S': 'SomeName'},
        'inventory': {'N': '500'},
        # ... more attributes ...
    }
)
```

A continuación se muestra la misma operación `PutItem` con la interfaz de recursos. La escritura de datos está implícita:

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')

table.put_item(
    Item={
        'pk': 'id#1',
        'sk': 'cart#123',
        'name': 'SomeName',
        'inventory': 500,
        # ... more attributes ...
    }
)
```

Si es necesario, puede realizar la conversión entre JSON normal y JSON de DynamoDB mediante las clases `TypeSerializer` y `TypeDeserializer` que se proporcionan con Boto3:

```
def dynamo_to_python(dynamo_object: dict) -> dict:
    deserializer = TypeDeserializer()
    return {
        k: deserializer.deserialize(v)
        for k, v in dynamo_object.items()
    }
```

```
def python_to_dynamo(python_object: dict) -> dict:
    serializer = TypeSerializer()
    return {
        k: serializer.serialize(v)
        for k, v in python_object.items()
    }
```

A continuación, se muestra cómo realizar una consulta desde la interfaz de cliente. La consulta se expresa como constructo de JSON. Utiliza una cadena `KeyConditionExpression` que requiere sustituir las variables para gestionar cualquier posible conflicto con las palabras clave:

```
import boto3

client = boto3.client('dynamodb')

# Construct the query
response = client.query(
    TableName='YourTableName',
    KeyConditionExpression='pk = :pk_val AND begins_with(sk, :sk_val)',
    FilterExpression='#name = :name_val',
    ExpressionAttributeValues={
        ':pk_val': {'S': 'id#1'},
        ':sk_val': {'S': 'cart#'},
        ':name_val': {'S': 'SomeName'},
    },
    ExpressionAttributeNames={
        '#name': 'name',
    }
)
```

La misma operación de consulta con la interfaz de recursos se puede acortar y simplificar:

```
import boto3
from boto3.dynamodb.conditions import Key, Attr

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('YourTableName')

response = table.query(
    KeyConditionExpression=Key('pk').eq('id#1') & Key('sk').begins_with('cart#'),
    FilterExpression=Attr('name').eq('SomeName')
)
```

Por último, imagine que desea saber el tamaño aproximado de una tabla (es decir, los metadatos que se guardan en la tabla y que se actualizan aproximadamente cada 6 horas). Con la interfaz de cliente, realiza una operación `describe_table()` y extrae la respuesta de la estructura JSON devuelta:

```
import boto3

dynamodb = boto3.client('dynamodb')

response = dynamodb.describe_table(TableName='YourTableName')
size = response['Table']['TableSizeBytes']
```

Con la interfaz de recursos, la tabla realiza la operación de descripción de forma implícita y presenta los datos directamente como un atributo:

```
import boto3

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')
size = table.table_size_bytes
```

> **Note**
>
> Al plantearse si desarrollar con la interfaz de cliente o de recursos, tenga en cuenta que no se añadirán nuevas características a la interfaz de recursos según se indica en la documentación del recurso: «The AWS Python SDK team does not intend to add new features to the resources interface in boto3. Existing interfaces will continue to operate during boto3's lifecycle. Customers can find access to newer service features through the client interface».

Uso del recurso de tabla batch_writer

`batch_writer` es una opción muy cómoda que solo está disponible en el recurso de tabla de nivel superior. DynamoDB admite operaciones de escritura por lotes, lo que permite realizar hasta 25 operaciones de inserción o eliminación en una solicitud de red. Este tipo de procesamiento por lotes mejora la eficiencia al minimizar los viajes de ida y vuelta de la red.

En la biblioteca cliente de bajo nivel, se utiliza la operación `client.batch_write_item()` para ejecutar lotes. Debe dividir manualmente el trabajo en lotes de 25. Después de cada operación, también debe solicitar que se envíe una lista de los elementos sin procesar (algunas de las operaciones de escritura pueden realizarse correctamente y otras pueden fallar). A continuación, debe volver a transferir los elementos no procesados a una operación `batch_write_item()` posterior. Hay una gran cantidad de código reutilizable.

El método Table.batch_writer crea un administrador de contexto para escribir objetos en un lote. Este método incluye una interfaz en la que parece que está escribiendo los elementos de uno en uno, pero internamente los almacena en búfer y los envía por lotes. También gestiona implícitamente los reintentos de los elementos no procesados.

```
dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('YourTableName')

movies = # long list of movies in {'pk': 'val', 'sk': 'val', etc} format
with table.batch_writer() as writer:
    for movie in movies:
        writer.put_item(Item=movie)
```

Ejemplos de código adicionales que exploran las capas de cliente y de recursos

También puede consultar los siguientes repositorios de ejemplos de código, que exploran el uso de las distintas funciones, tanto con el cliente como con el recurso:

- Ejemplos de código oficiales de una sola acción de AWS
- Ejemplos de código oficiales orientados a escenarios de AWS
- Ejemplos de código de una sola acción actualizados por la comunidad

Comprensión sobre la interacción de los objetos Client y Resource con las sesiones y los subprocesos

El objeto Resource no es seguro para los subprocesos y no debe compartirse entre subprocesos o procesos. Consulte la guía sobre Resource para obtener más información.

El objeto Client, por el contrario, generalmente es seguro para los subprocesos, excepto en el caso de las características avanzadas específicas. Consulte la guía sobre Clients para obtener más información.

El objeto Session no es seguro para los subprocesos. Por lo tanto, cada vez que cree un Client o Resource en un entorno con varios subprocesos, primero debe crear una nueva Session y, a continuación, crear el Client o el Resource a partir de la Session. Consulte la guía sobre Sessions para obtener más información.

Cuando llama al `boto3.resource()`, está utilizando la Session predeterminada de forma implícita. Esto resulta práctico para escribir código de un solo subproceso. Al escribir código con varios subprocesos, primero deberá construir una nueva Session para cada subproceso y, a continuación, recuperar el recurso de esa Session:

```
# Explicitly create a new Session for this thread
session = boto3.Session()
dynamodb = session.resource('dynamodb')
```

Personalización del objeto Config

Al crear un objeto Client o Resource, puede transferir parámetros con nombres opcionales para personalizar el comportamiento. El parámetro denominado `config` desbloquea una variedad de funciones. Se trata de una instancia `botocore.client.Config` y la documentación de referencia de Config muestra todo lo que expone y que debe controlar. La guía de Configuration ofrece una buena descripción general.

> **Note**
>
> Puede modificar muchos de estos ajustes de comportamiento en el nivel de Session, en el archivo de configuración de AWS o como variables de entorno.

Config para tiempos de espera

Uno de los usos de una configuración personalizada es ajustar los comportamientos de la red:

- connect_timeout (flotante o entero): tiempo en segundos que transcurre hasta que se produzca una excepción de tiempo de espera al intentar establecer una conexión. El valor predeterminado es de 60 segundos.

- read_timeout (flotante o entero): tiempo en segundos que transcurre hasta que se produzca una excepción de tiempo de espera al intentar leer una conexión. El valor predeterminado es de 60 segundos.

Los tiempos de espera de 60 segundos son excesivos para DynamoDB. Esto significa que un fallo transitorio en la red provocará un minuto de retardo hasta que el cliente pueda volver a intentarlo. El siguiente código acorta los tiempos de espera hasta un segundo:

```
import boto3
from botocore.config import Config

my_config = Config(
   connect_timeout = 1.0,
   read_timeout = 1.0
)
dynamodb = boto3.resource('dynamodb', config=my_config)
```

Para obtener más información sobre los tiempos de espera, consulte Tuning AWS Java SDK HTTP request settings for latency-aware DynamoDB applications. Tenga en cuenta que el SDK de Java tiene más configuraciones de tiempo de espera que Python.

Config para keep-alive

Si utiliza botocore 1.27.84 o posterior, también puede controlar el TCP Keep-Alive:

- tcp_keepalive (booleano): activa la opción de conector TCP Keep-Alive que se usa al crear nuevas conexiones si está configurada en `True` (el valor predeterminado es `False`). Solo está disponible a partir de la versión 1.27.84 de botocore.

Si se configura TCP Keep-Alive en `True`, se pueden reducir las latencias medias. Este es un ejemplo de código que establece el valor de TCP Keep-Alive condicionalmente en true si tiene la versión de botocore correcta:

```
import botocore
import boto3
from botocore.config import Config
from distutils.version import LooseVersion

required_version = "1.27.84"
current_version = botocore.__version__
```

```
my_config = Config(
   connect_timeout = 0.5,
   read_timeout = 0.5
)
if LooseVersion(current_version) > LooseVersion(required_version):
    my_config = my_config.merge(Config(tcp_keepalive = True))

dynamodb = boto3.resource('dynamodb', config=my_config)
```

> **Note**
>
> TCP Keep-Alive no es lo mismo que HTTP Keep-Alive. Con TCP Keep-Alive, el sistema operativo subyacente envía paquetes pequeños a través de la conexión de socket para mantener la conexión activa y detectar inmediatamente cualquier caída. Con HTTP Keep-Alive, se reutiliza la conexión web integrada en el socket subyacente. HTTP Keep-Alive siempre está activado en boto3.

Hay un límite en cuanto al tiempo que se puede mantener activa una conexión inactiva. Considere la posibilidad de enviar solicitudes periódicas, tal vez cada minuto, si tiene una conexión inactiva pero quiere que la siguiente solicitud utilice una conexión ya establecida.

Config para reintentos

La configuración también acepta un diccionario llamado reintentos en el que puede especificar el comportamiento de reintentos que espera. Los reintentos se producen dentro del SDK cuando este recibe un error de tipo transitorio. Si se vuelve a intentar un error internamente (y el reintento genera una respuesta correcta), desde el punto de vista del código, no se trata de ningún error, sino de una latencia ligeramente elevada. Estos son los valores que puede especificar:

- max_attempts: número entero que representa el número máximo de reintentos que se realizará en una sola solicitud. Por ejemplo, si establece este valor en 2, la solicitud se volverá a intentar como máximo dos veces después de la solicitud inicial. Si se establece en 0, no se intentará de nuevo después de la solicitud inicial.
- total_max_attempts: número entero que representa el número total máximo de intentos que se realizará en una sola solicitud. Esto incluye la solicitud inicial, por lo que un valor de 1 indica que no se volverá a intentar ninguna solicitud. Si se proporcionan `total_max_attempts` y `max_attempts`, `total_max_attempts` prevalece. Se prefiere `total_max_attempts` sobre

`max_attempts` porque se asigna a la variable de entorno de `AWS_MAX_ATTEMPTS` y al valor del archivo de configuración `max_attempts`.

- mode: cadena que representa el tipo de modo de reintento que debe utilizar botocore. Los valores válidos son:
  - legacy: modo predeterminado. Espera 50 ms en el primer reintento y, a continuación, utiliza un retroceso exponencial con un factor base de 2. En el caso de DynamoDB, realiza un máximo de 10 intentos, a menos que se anule con lo anterior.

    > **Note**
    >
    > Con un retroceso exponencial, el último intento esperará casi 13 segundos.

  - standard: se denomina estándar porque es más coherente con otros SDK de AWS. Este modo espera un período de tiempo aleatorio que va desde 0 ms hasta 1000 ms para el primer reintento. Si es necesario volver a intentarlo, selecciona otro tiempo aleatorio de 0 ms hasta 1000 ms y lo multiplica por 2. Si es necesario volver a intentarlo, realiza la misma selección aleatoria multiplicada por 4, y así sucesivamente. Cada espera tiene un límite de 20 segundos. Este modo realizará reintentos en caso de que se detecten más condiciones de fallo que el modo `legacy`. En el caso de DynamoDB, realiza un máximo de 3 intentos (a menos que se anule con lo anterior).
  - adaptive: modo de reintento experimental que incluye la funcionalidad del modo estándar, pero añade una limitación automática del lado del cliente. Con la limitación de velocidad adaptativa, los SDK pueden ralentizar la velocidad a la que se envían las solicitudes para adaptarse mejor a la capacidad de los servicios de AWS. Se trata de un modo provisional cuyo comportamiento puede cambiar.

La definición ampliada de estos modos de reintento está disponible en la guía de reintentos, así como en el tema sobre el comportamiento de los reintentos de la referencia del SDK.

Este es un ejemplo en el que se utiliza explícitamente la política de reintentos `legacy` con un máximo de 3 solicitudes en total (2 reintentos):

```
import boto3
from botocore.config import Config

my_config = Config(
   connect_timeout = 1.0,
   read_timeout = 1.0,
```

```
    retries = {
      'mode': 'legacy',
      'total_max_attempts': 3
    }
)
dynamodb = boto3.resource('dynamodb', config=my_config)
```

Dado que DynamoDB es un sistema de alta disponibilidad y baja latencia, puede ser más agresivo con la velocidad de los reintentos de lo que permiten las políticas de reintentos integradas. Puede implementar su propia política de reintentos estableciendo el número máximo de intentos en 0, detectando usted mismo las excepciones y reintentándolo según convenga con su propio código, en lugar de depender de boto3 para realizar los reintentos implícitos.

Si administra su propia política de reintentos, se recomienda diferenciar entre limitaciones y errores:

- Una limitación (indicada con una `ProvisionedThroughputExceededException` o `ThrottlingException`) representa un servicio en buen estado que le informa de que ha superado la capacidad de lectura o escritura en una tabla o partición de DynamoDB. Cada milisegundo que pasa, se dispone de un poco más de capacidad de lectura o escritura, por lo que puede volver a intentarlo rápidamente (por ejemplo, cada 50 ms) para intentar acceder a la capacidad recién liberada. Con las limitaciones no se requiere un retroceso exponencial, pues las limitaciones son ligeras para que DynamoDB las devuelva. Además, no se le cobra por solicitud. El retroceso exponencial asigna retardos más prolongados a los subprocesos de los clientes que ya han esperado más tiempo, lo que, estadísticamente, prolonga la p50 y la p99 hacia fuera.
- Un error (identificado con `InternalServerError` o `ServiceUnavailable`, entre otros) indica un problema transitorio con el servicio. Esto puede ser para toda la tabla o, posiblemente, solo para la partición desde la que está leyendo o escribiendo. En el caso de los errores, puede hacer una pausa más larga antes de volver a intentarlo, por ejemplo, 250 ms o 500 ms, y utilizar la fluctuación para escalonar los reintentos.

Config para un máximo de conexiones de grupo

Por último, la configuración le permite controlar el tamaño del grupo de conexiones:

- max_pool_connections (entero): número máximo de conexiones que se debe mantener en un grupo de conexiones. Si no se especifica ningún valor, se utiliza el valor predeterminado 10.

Esta opción controla el número máximo de conexiones HTTP que se debe mantener agrupado para volverlas a utilizar. Se conserva un grupo diferente por sesión. Si prevé que se van a dirigir más de 10 subprocesos a los clientes o los recursos creados en la misma sesión, debería plantearse aumentarlo para que los subprocesos no tengan que esperar a que otros subprocesos utilicen una conexión agrupada.

```
import boto3
from botocore.config import Config

my_config = Config(
   max_pool_connections = 20
)

# Setup a single session holding up to 20 pooled connections
session = boto3.Session(my_config)

# Create up to 20 resources against that session for handing to threads
# Notice the single-threaded access to the Session and each Resource
resource1 = session.resource('dynamodb')
resource2 = session.resource('dynamodb')
# etc
```

Gestión de errores

No todas las excepciones de servicio de AWS se definen estáticamente en Boto3. Esto se debe a que los errores y las excepciones de los servicios de AWS varían ampliamente y están sujetos a cambios. Boto3 agrupa todas las excepciones del servicio como un `ClientError` y expone los detalles como un JSON estructurado. Por ejemplo, una respuesta de error podría estructurarse de la siguiente manera:

```
{
    'Error': {
        'Code': 'SomeServiceException',
        'Message': 'Details/context around the exception or error'
    },
    'ResponseMetadata': {
        'RequestId': '1234567890ABCDEF',
        'HostId': 'host ID data will appear here as a hash',
        'HTTPStatusCode': 400,
        'HTTPHeaders': {'header metadata key/values will appear here'},
        'RetryAttempts': 0
```

```
    }
}
```

El código siguiente detecta cualquier excepción `ClientError` y examina el valor de la cadena de `Code` dentro del `Error` para determinar qué acción se debe realizar:

```
import botocore
import boto3

dynamodb = boto3.client('dynamodb')

try:
    response = dynamodb.put_item(...)

except botocore.exceptions.ClientError as err:
    print('Error Code: {}'.format(err.response['Error']['Code']))
    print('Error Message: {}'.format(err.response['Error']['Message']))
    print('Http Code: {}'.format(err.response['ResponseMetadata']['HTTPStatusCode']))
    print('Request ID: {}'.format(err.response['ResponseMetadata']['RequestId']))

    if err.response['Error']['Code'] in ('ProvisionedThroughputExceededException',
 'ThrottlingException'):
        print("Received a throttle")
    elif err.response['Error']['Code'] == 'InternalServerError':
        print("Received a server error")
    else:
        raise err
```

Algunos códigos de excepción (pero no todos) se han materializado como clases de nivel superior. Puede optar por gestionarlos directamente. Cuando se utiliza la interfaz Client, estas excepciones se rellenan dinámicamente en el cliente y se detectan mediante la instancia de cliente, de la siguiente manera:

```
except ddb_client.exceptions.ProvisionedThroughputExceededException:
```

Cuando utilice la interfaz Resource, tendrá que usar `.meta.client` para ir desde el recurso hasta el cliente subyacente para acceder a las excepciones, de la siguiente manera:

```
except ddb_resource.meta.client.exceptions.ProvisionedThroughputExceededException:
```

Puede generar la lista de tipos de excepciones materializadas de forma dinámica para consultarla:

```
ddb = boto3.client("dynamodb")
print([e for e in dir(ddb.exceptions) if e.endswith('Exception') or
 e.endswith('Error')])
```

Al realizar una operación de escritura con una expresión de condición, puede solicitar que, si la expresión falla, se devuelva el valor del elemento en la respuesta de error.

```
try:
    response = table.put_item(
        Item=item,
        ConditionExpression='attribute_not_exists(pk)',
        ReturnValuesOnConditionCheckFailure='ALL_OLD'
    )
except table.meta.client.exceptions.ConditionalCheckFailedException as e:
    print('Item already exists:', e.response['Item'])
```

Para obtener más información sobre la gestión de errores y las excepciones:

- La guía de boto3 sobre la gestión de errores contiene más información sobre las técnicas de gestión de errores.
- En la sección sobre errores de programación de la guía para desarrolladores de DynamoDB se enumeran los errores que pueden producirse.
- La sección Common Errors de la referencia de la API.
- En la documentación de cada operación de la API se enumeran los errores que puede generar esa llamada (por ejemplo, BatchWriteItem).

Registro

La biblioteca boto3 se integra con el módulo de registro incluido de Python para hacer un seguimiento de lo que ocurre durante una sesión. Para controlar los niveles de registro, puede configurar el módulo de registro:

```
import logging

logging.basicConfig(level=logging.INFO)
```

Esto configura el registrador raíz para registrar `INFO` y los mensajes de nivel superior. Se ignorarán los mensajes de registro que sean menos graves que el nivel. Los niveles de registro son `DEBUG`, `INFO`, `WARNING`, `ERROR` y `CRITICAL`. El valor predeterminado es `WARNING`.

Los registradores en boto3 son jerárquicos. La biblioteca usa varios registradores diferentes y cada uno equivale a diferentes partes de la biblioteca. Puede controlar el comportamiento de cada uno de ellos por separado:

- boto3: es el registrador principal del módulo boto3.
- botocore: es el registrador principal del paquete botocore.
- botocore.auth: se utiliza para registrar la creación de firmas de AWS para las solicitudes.
- botocore.credentials: se utiliza para registrar el proceso de obtención y actualización de credenciales.
- botocore.endpoint: se utiliza para registrar la creación de solicitudes antes de enviarlas a través de la red.
- botocore.hooks: se utiliza para los eventos de registro desencadenados en la biblioteca.
- botocore.loaders: se utiliza para registrar la carga de partes de los modelos de servicio de AWS.
- botocore.parsers: se utiliza para registrar las respuestas del servicio de AWS antes de analizarlas.
- botocore.retryhandler: se utiliza para registrar el procesamiento de los reintentos de solicitudes de servicio de AWS (modo heredado).
- botocore.retryhandler: se utiliza para registrar el procesamiento de los reintentos de solicitudes de servicio de AWS (modo estándar o adaptativo).
- botocore.utils: se utiliza para registrar diversas actividades en la biblioteca.
- botocore.waiter: se utiliza para registrar la funcionalidad de los esperadores, que consultan un servicio de AWS hasta que se alcance un estado determinado.

Otras bibliotecas también registran. Internamente, boto3 utiliza la urllib3 de terceros para gestionar la conexión HTTP. Si la latencia es importante, puede observar sus registros para asegurarse de que su grupo se utiliza correctamente. Para ello, comprueba cuándo urllib3 establece una nueva conexión o cierra una que está inactiva.

- urllib3.connectionpool: se utiliza para registrar los eventos de gestión de grupos de conexiones.

El siguiente fragmento de código establece la mayoría de los registros `INFO` con el registro `DEBUG` para el punto de conexión y la actividad del grupo de conexiones:

```
import logging

logging.getLogger('boto3').setLevel(logging.INFO)
logging.getLogger('botocore').setLevel(logging.INFO)
logging.getLogger('botocore.endpoint').setLevel(logging.DEBUG)
logging.getLogger('urllib3.connectionpool').setLevel(logging.DEBUG)
```

Enlaces de eventos

Botocore emite eventos durante varias partes de su ejecución. Puede registrar controladores para estos eventos de modo que cada vez que se emita un evento, se llame a su controlador. Esto le permite ampliar el comportamiento de botocore sin tener que modificar las partes internas.

Por ejemplo, supongamos que quiere saber todas las veces que se llama a una operación `PutItem` en cualquier tabla de DynamoDB de su aplicación. Puede registrarse en el evento `'provide-client-params.dynamodb.PutItem'` para detectar y registrar cada vez que se invoque una operación `PutItem` en la sesión asociada. A continuación se muestra un ejemplo:

```
import boto3
import botocore
import logging

def log_put_params(params, **kwargs):
    if 'TableName' in params and 'Item' in params:
        logging.info(f"PutItem on table {params['TableName']}: {params['Item']}")

logging.basicConfig(level=logging.INFO)

session = boto3.Session()
event_system = session.events

# Register our interest in hooking in when the parameters are provided to PutItem
event_system.register('provide-client-params.dynamodb.PutItem', log_put_params)

# Now, every time you use this session to put an item in DynamoDB,
# it will log the table name and item data.
dynamodb = session.resource('dynamodb')
table = dynamodb.Table('YourTableName')
table.put_item(
    Item={
        'pk': '123',
        'sk': 'cart#123',
```

```
        'item_data': 'YourItemData',
        # ... more attributes ...
    }
)
```

Dentro del controlador puede incluso manipular los parámetros mediante programación para cambiar el comportamiento:

```
params['TableName'] = "NewTableName"
```

Para obtener más información sobre los eventos, consulte la documentación de botocore sobre eventos y la documentación de boto3 sobre eventos.

La paginación y el paginador

Algunas solicitudes, como Query y Scan, limitan el tamaño de los datos que se devuelven en una sola solicitud y requieren que realice solicitudes repetidas para abrir las páginas siguientes.

Puede controlar el número máximo de elementos que se debe leer en cada página con el parámetro `limit`. Por ejemplo, puede usar `limit` para recuperar solo los últimos 10 elementos. Tenga en cuenta que este límite indica cuánto debe leerse de la tabla antes de aplicar un filtro. No hay forma de especificar que quiere exactamente 10 después de filtrar; solo puede controlar el recuento previo al filtro y comprobarlo del lado del cliente cuando haya obtenido realmente 10. Cada respuesta tiene siempre un tamaño máximo de 1 MB, sin importar el límite.

Si la respuesta incluye una `LastEvaluatedKey`, significa que la respuesta ha finalizado porque ha alcanzado un límite de recuento o tamaño. Esta clave es la última clave evaluada para esa respuesta. Puede recuperar esta `LastEvaluatedKey` y pasarla a una llamada de seguimiento como `ExclusiveStartKey` para que lea el siguiente fragmento desde ese punto de partida. Si no se devuelve ninguna `LastEvaluatedKey`, significa que no hay más elementos que coincidan con la consulta o el análisis.

Este es un ejemplo sencillo (con la interfaz de recursos, pero la interfaz de cliente sigue el mismo patrón) en el que se leen como máximo 100 elementos por página y se repite hasta que se hayan leído todos los elementos.

```
import boto3

dynamodb = boto3.resource('dynamodb')
```

```
table = dynamodb.Table('YourTableName')

query_params = {
    'KeyConditionExpression': Key('pk').eq('123') & Key('sk').gt(1000),
    'Limit': 100
}

while True:
    response = table.query(**query_params)

    # Process the items however you like
    for item in response['Items']:
        print(item)

    # No LastEvaluatedKey means no more items to retrieve
    if 'LastEvaluatedKey' not in response:
        break

    # If there are possibly more items, update the start key for the next page
    query_params['ExclusiveStartKey'] = response['LastEvaluatedKey']
```

boto3 puede hacerlo por usted con la opción Paginators. Sin embargo, solo funciona con la interfaz Client. Este es el código reescrito para usar Paginators:

```
import boto3

dynamodb = boto3.client('dynamodb')

paginator = dynamodb.get_paginator('query')

query_params = {
    'TableName': 'YourTableName',
    'KeyConditionExpression': 'pk = :pk_val AND sk > :sk_val',
    'ExpressionAttributeValues': {
        ':pk_val': {'S': '123'},
        ':sk_val': {'N': '1000'},
    },
    'Limit': 100
}

page_iterator = paginator.paginate(**query_params)

for page in page_iterator:
```

```
    # Process the items however you like
    for item in page['Items']:
        print(item)
```

Para obtener más información, consulte la guía sobre Paginators y la referencia de la API para DynamoDB.Paginator.Query.

> **Note**
>
> Paginators también tiene sus propias opciones de configuración denominadas `MaxItems`, `StartingToken` y `PageSize`. Para paginar con DynamoDB, debe ignorar esta configuración.

Esperadores

Los esperadores ofrecen la posibilidad de esperar a que se complete algo antes de continuar. Por el momento, solo permiten que se espere hasta que se cree o elimine una tabla. En segundo plano, el esperador realiza una comprobación cada 20 segundos hasta 25 veces. Puede hacerlo usted mismo, pero usar un esperador es más elegante a la hora de escribir de forma automática.

Este código muestra cómo esperar a que se cree una tabla en particular:

```
# Create a table, wait until it exists, and print its ARN
response = client.create_table(...)
waiter = client.get_waiter('table_exists')
waiter.wait(TableName='YourTableName')
print('Table created:', response['TableDescription']['TableArn']
```

Para obtener más información, consulte la guía sobre Waiters y la referencia sobre Waiters.

Programación de Amazon DynamoDB con JavaScript

Esta guía ofrece orientación a los programadores que desean utilizar Amazon DynamoDB con JavaScript. Obtenga información sobre los AWS SDK para JavaScript, las capas de abstracción disponibles, la configuración de las conexiones, la gestión de errores, la definición de las políticas de reintentos, la gestión del mantenimiento activo y mucho más.

Temas

- Acerca de AWS SDK para JavaScript
- Uso del AWS SDK para JavaScript V3
- Acceso a la documentación de JavaScript
- Capas de abstracción
- Uso de la función de utilidad de serialización
- Lectura de elementos
- Escrituras condicionales
- Paginación
- Especificación de la configuración
- Esperadores
- Gestión de errores
- Registro
- Consideraciones

Acerca de AWS SDK para JavaScript

El AWS SDK para JavaScript proporciona acceso a los Servicios de AWS que usan scripts de navegador o Node.js. Esta documentación se centra en la versión más reciente del SDK (V3). AWS actualiza el AWS SDK para JavaScript V3 como un proyecto de código abierto alojado en GitHub. Los problemas y las solicitudes de características son públicos. Puede acceder a ellos desde la página de problemas del repositorio de GitHub.

JavaScript V2 es similar a V3, pero incluye algunas diferencias de sintaxis. La V3 es más modular, lo que facilita el envío de dependencias más pequeñas, y tiene una compatibilidad con TypeScript de primer nivel. Recomendamos el uso de la versión más reciente del SDK.

Uso del AWS SDK para JavaScript V3

Puede añadir el SDK a su aplicación Node.js mediante el administrador de paquetes de Node. En los ejemplos siguientes se muestra cómo añadir los paquetes de SDK más habituales para trabajar con DynamoDB.

- `npm install @aws-sdk/client-dynamodb`
- `npm install @aws-sdk/lib-dynamodb`

- `npm install @aws-sdk/util-dynamodb`

Al instalar los paquetes, se añaden referencias a la sección de dependencias del archivo de proyecto package.json. Tiene la opción de usar la sintaxis del módulo ECMAScript más reciente. Para obtener más información sobre estos dos enfoques, consulte la sección Consideraciones.

Acceso a la documentación de JavaScript

Comience a utilizar la documentación de JavaScript con los siguientes recursos:

- Acceda a la guía para desarrolladores para ver la documentación básica de JavaScript. Las instrucciones de instalación están disponibles en la sección Configuración.
- Acceda a la documentación de la referencia de la API para explorar todas las clases y métodos disponibles.
- El SDK para JavaScript es compatible con muchos otros Servicios de AWS además de DynamoDB. Utilice el siguiente procedimiento para localizar la cobertura de la API específica para DynamoDB:

  1. En Services, elija DynamoDB and Libraries. Esto documenta el cliente de bajo nivel.
  2. Elija lib-dynamodb. Esto documenta el cliente de alto nivel. Los dos clientes representan dos capas de abstracción distintas que puede utilizar. Consulte la sección siguiente para obtener más información sobre las capas de abstracción.

Capas de abstracción

El SDK para JavaScript V3 tiene un cliente de bajo nivel (`DynamoDBClient`) y un cliente de alto nivel (`DynamoDBDocumentClient`).

Temas

- Cliente de bajo nivel (DynamoDBClient)
- Cliente de alto nivel (DynamoDBDocumentClient)

Cliente de bajo nivel (**`DynamoDBClient`**)

El cliente de bajo nivel no proporciona abstracciones adicionales del protocolo de conexión subyacente. Este cliente le da control total sobre todos los aspectos de la comunicación, pero como

no hay abstracciones, debe hacer cosas como proporcionar definiciones de elementos con el formato JSON de DynamoDB.

Tal como se muestra en el siguiente ejemplo, con este formato los tipos de datos deben indicarse de forma explícita. La S hace referencia a un valor de cadena y la N, a un valor numérico. Los números de la conexión siempre se envían como cadenas etiquetadas como tipos numéricos para garantizar que no se pierda precisión. Las llamadas a la API de bajo nivel tienen un patrón de nomenclatura como `PutItemCommand` y `GetItemCommand`.

En el siguiente ejemplo, se utiliza un cliente de bajo nivel con `Item` definido con el JSON de DynamoDB:

```
const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function addProduct() {
  const params = {
    TableName: "products",
    Item: {
      "id": { S: "Product01" },
      "description": { S: "Hiking Boots" },
      "category": { S: "footwear" },
      "sku": { S: "hiking-sku-01" },
      "size": { N: "9" }
    }
  };

  try {
    const data = await client.send(new PutItemCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}
addProduct();
```

Cliente de alto nivel (**`DynamoDBDocumentClient`**)

El cliente de documentos de alto nivel de DynamoDB integra unas características muy prácticas, por ejemplo, elimina la necesidad de serializar los datos manualmente, y permite la lectura y escritura

directas con objetos JavaScript estándar. En la documentación para `lib-dynamodb` se incluye la lista de ventajas.

Para crear una instancia de `DynamoDBDocumentClient`, cree un `DynamoDBClient` de bajo nivel y, a continuación, encapsúlelo con un `DynamoDBDocumentClient`. La convención de nomenclatura de las funciones difiere ligeramente entre los dos paquetes. Por ejemplo, el bajo nivel usa `PutItemCommand`, mientras que el alto nivel usa `PutCommand`. Los distintos nombres permiten que ambos conjuntos de funciones cohabiten en el mismo contexto, lo que permite mezclar ambos en el mismo script.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const { DynamoDBDocumentClient, PutCommand } = require("@aws-sdk/lib-dynamodb");

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function addProduct() {
  const params = {
    TableName: "products",
    Item: {
      id: "Product01",
      description: "Hiking Boots",
      category: "footwear",
      sku: "hiking-sku-01",
      size: 9,
    },
  };

  try {
    const data = await docClient.send(new PutCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}

addProduct();
```

El patrón de uso es coherente cuando se leen elementos con operaciones de la API como `GetItem`, `Query` o `Scan`.

Uso de la función de utilidad de serialización

Puede utilizar el cliente de bajo nivel y serializar o deserializar los tipos de datos por su cuenta. El paquete de utilidades, util-dynamodb, tiene una función de utilidad `marshall()` que acepta JSON y produce JSON de DynamoDB, así como una función `unmarshall()` que hace lo contrario. En el siguiente ejemplo, se utiliza el cliente de bajo nivel y la llamada a `marshall()` gestiona la serialización de los datos.

```
const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");
const { marshall } = require("@aws-sdk/util-dynamodb");

const client = new DynamoDBClient({});

async function addProduct() {
  const params = {
    TableName: "products",
    Item: marshall({
      id: "Product01",
      description: "Hiking Boots",
      category: "footwear",
      sku: "hiking-sku-01",
      size: 9,
    }),
  };

  try {
    const data = await client.send(new PutItemCommand(params));
  } catch (error) {
    console.error("Error:", error);
  }
}
addProduct();
```

Lectura de elementos

Para leer un solo elemento de DynamoDB, se utiliza la operación de la API `GetItem`. De igual modo que con el comando `PutItem`, tiene la opción de utilizar el cliente de bajo nivel o el cliente de documentos de alto nivel. En el siguiente ejemplo se muestra el uso del cliente de documentos de alto nivel para recuperar un elemento.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
```

```
const { DynamoDBDocumentClient, GetCommand } = require("@aws-sdk/lib-dynamodb");

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function getProduct() {
  const params = {
    TableName: "products",
    Key: {
      id: "Product01",
    },
  };

  try {
    const data = await docClient.send(new GetCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}

getProduct();
```

Utilice la operación de la API `Query` para leer varios elementos. Puede usar el cliente de bajo nivel o el cliente de documentos. En el siguiente ejemplo se utiliza el cliente de documentos de alto nivel.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  QueryCommand,
} = require("@aws-sdk/lib-dynamodb");

const client = new DynamoDBClient({});

const docClient = DynamoDBDocumentClient.from(client);

async function productSearch() {
  const params = {
    TableName: "products",
    IndexName: "GSI1",
    KeyConditionExpression: "#category = :category and begins_with(#sku, :sku)",
    ExpressionAttributeNames: {
```

```
      "#category": "category",
      "#sku": "sku",
    },
    ExpressionAttributeValues: {
      ":category": "footwear",
      ":sku": "hiking",
    },
  };

  try {
    const data = await docClient.send(new QueryCommand(params));
    console.log('result : ' + JSON.stringify(data));
  } catch (error) {
    console.error("Error:", error);
  }
}

productSearch();
```

Escrituras condicionales

Las operaciones de escritura de DynamoDB pueden especificar una expresión de condición lógica que debe evaluarse como verdadera para que la escritura continúe. Si la evaluación de condición no es verdadera, la operación de escritura genera una excepción. La expresión de condición puede comprobar si el elemento ya existe o si sus atributos coinciden con determinadas restricciones.

`ConditionExpression = "version = :ver AND size(VideoClip) < :maxsize"`

Si se produce un error en la expresión de condición, puede usar `ReturnValuesOnConditionCheckFailure` para solicitar que la respuesta al error incluya el elemento que no cumplía las condiciones para deducir cuál era el problema. Para obtener más información, consulte Handle conditional write errors in high concurrency scenarios with Amazon DynamoDB.

```
try {
      const response = await client.send(new PutCommand({
          TableName: "YourTableName",
          Item: item,
          ConditionExpression: "attribute_not_exists(pk)",
          ReturnValuesOnConditionCheckFailure: "ALL_OLD"
      }));
  } catch (e) {
```

```
    if (e.name === 'ConditionalCheckFailedException') {
        console.log('Item already exists:', e.Item);
    } else {
        throw e;
    }
}
```

Los ejemplos de código adicionales que muestran otros aspectos del uso de JavaScript SDK V3 están disponibles en la documentación de JavaScript SDK V3 y en el repositorio de GitHub de DynamoDB-SDK-Examples.

Paginación

Temas

- Uso del método de conveniencia paginateScan

Las solicitudes de lectura como `Scan` y `Query` es probable que devuelvan varios elementos de un conjunto de datos. Si realiza un `Scan` o `Query` con un parámetro `Limit`, una vez que el sistema haya leído todos esos elementos, se enviará una respuesta parcial y tendrá que paginar para recuperar los elementos adicionales.

El sistema solo leerá un máximo de 1 megabyte de datos por solicitud. Si incluye una expresión `Filter`, el sistema seguirá leyendo un megabyte, como máximo, de datos del disco, pero devolverá los elementos de ese megabyte que coincidan con el filtro. La operación de filtrado podría devolver 0 elementos para una página, pero requerir una paginación adicional antes de que se agote la búsqueda.

Debe buscar la `LastEvaluatedKey` en la respuesta y usarla como parámetro `ExclusiveStartKey` en una solicitud posterior para continuar con la recuperación de datos. Esto sirve como marcador, tal como se indica en el siguiente ejemplo.

> **Note**
>
> El ejemplo pasa una `lastEvaluatedKey` nula como la `ExclusiveStartKey` de la primera iteración. Esto está permitido.

Ejemplo de uso de la `LastEvaluatedKey`:

```
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function paginatedScan() {
  let lastEvaluatedKey;
  let pageCount = 0;

  do {
    const params = {
      TableName: "products",
      ExclusiveStartKey: lastEvaluatedKey,
    };

    const response = await client.send(new ScanCommand(params));
    pageCount++;
    console.log(`Page ${pageCount}, Items:`, response.Items);
    lastEvaluatedKey = response.LastEvaluatedKey;
  } while (lastEvaluatedKey);
}

paginatedScan().catch((err) => {
  console.error(err);
});
```

Uso del método de conveniencia **`paginateScan`**

El SDK proporciona métodos de conveniencia denominados `paginateScan` y `paginateQuery` que hacen que este trabajo funcione y realiza las solicitudes repetidas entre bastidores. Especifique el número máximo de elementos que se van a leer por solicitud mediante el parámetro estándar `Limit`.

```
const { DynamoDBClient, paginateScan } = require("@aws-sdk/client-dynamodb");

const client = new DynamoDBClient({});

async function paginatedScanUsingPaginator() {
  const params = {
    TableName: "products",
    Limit: 100
  };
```

```
  const paginator = paginateScan({client}, params);

  let pageCount = 0;

  for await (const page of paginator) {
    pageCount++;
    console.log(`Page ${pageCount}, Items:`, page.Items);
  }
}

paginatedScanUsingPaginator().catch((err) => {
  console.error(err);
});
```

> **Note**
>
> El análisis regular de toda la tabla no es un patrón de acceso recomendado, a menos que la tabla sea pequeña.

Especificación de la configuración

Temas

- Config para tiempos de espera
- Config para keep-alive
- Config para reintentos

Al configurar el `DynamoDBClient`, puede especificar varias anulaciones de configuración transfiriendo un objeto de configuración al constructor. Por ejemplo, puede especificar la región a la que desea conectarse si el contexto de llamada o la URL del punto de conexión que desea utilizar todavía son desconocidas. Esto resulta útil si desea dirigirse a una instancia local de DynamoDB con fines de desarrollo.

```
const client = new DynamoDBClient({
  region: "eu-west-1",
  endpoint: "http://localhost:8000",
});
```

Config para tiempos de espera

DynamoDB usa HTTPS para la comunicación entre cliente y servidor. Puede controlar algunos aspectos de la capa HTTP proporcionando un objeto `NodeHttpHandler`. Por ejemplo, puede ajustar los valores de tiempo de espera clave `connectionTimeout` y `requestTimeout`. `connectionTimeout` es el tiempo máximo, en milisegundos, que el cliente esperará mientras intenta establecer una conexión antes de darse por vencido.

`requestTimeout` define cuánto tiempo esperará el cliente a recibir una respuesta después de que se envíe una solicitud, también en milisegundos. El valor predeterminado para ambos es cero, lo que significa que el tiempo de espera está desactivado y no hay límite en cuanto al tiempo que el cliente esperará si la respuesta no llega. Debe establecer los tiempos de espera en un valor razonable para que, en caso de un problema con la red, se produzca un error en la solicitud y se pueda lanzar una nueva solicitud. Por ejemplo:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";

const requestHandler = new NodeHttpHandler({
  connectionTimeout: 2000,
  requestTimeout: 2000,
});

const client = new DynamoDBClient({
  requestHandler
});
```

> **Note**
>
> En el ejemplo que se proporciona se utiliza la importación Smithy. Smithy es un lenguaje de código abierto actualizado por AWS para definir servicios y SDK.

Además de configurar los valores de tiempo de espera, puede establecer el número máximo de conexiones, lo que permite aumentar el número de conexiones simultáneas por origen. La guía para desarrolladores incluye detalles sobre la configuración del parámetro `maxSockets`.

Config para keep-alive

Cuando se usa HTTPS, la primera solicitud siempre requiere una comunicación de ida y vuelta para establecer una conexión segura. HTTP Keep-Alive permite que las solicitudes posteriores reutilicen la conexión ya establecida, lo que las hace más eficientes y reduce la latencia. HTTP Keep-Alive está activado de forma predeterminada con JavaScript V3.

Hay un límite en cuanto al tiempo que se puede mantener activa una conexión inactiva. Considere la posibilidad de enviar solicitudes periódicas, tal vez cada minuto, si tiene una conexión inactiva pero quiere que la siguiente solicitud utilice una conexión ya establecida.

> **Note**
>
> Tenga en cuenta que en la versión anterior del SDK, keep-alive estaba desactivado de forma predeterminada, lo que significaba que se cerraba cada conexión inmediatamente después usarla. Si usa la V2, puede anular esta configuración.

Config para reintentos

Cuando el SDK reciba una respuesta de error y el error se pueda reanudar según lo que determine el SDK, como una excepción de limitación o una excepción de servicio temporal, volverá a intentarlo. Al ser usted quien llama, no se da cuenta de nada, excepto si nota que la solicitud ha tardado más en tramitarse.

El SDK para JavaScript V3 realizará un total de tres solicitudes de forma predeterminada antes de darse por vencido y pasar el error al contexto de la llamada. Puede ajustar el número y la frecuencia de estos reintentos.

El constructor `DynamoDBClient` acepta una configuración `maxAttempts` que limita el número de intentos que se realizará. En el siguiente ejemplo se incrementa el valor predeterminado de 3 a un total de 5. Si lo establece en 0 o 1, significa que no quiere ningún reintento automático y que usted quiere gestionar cualquier error reanudable dentro del bloque Catch.

```
const client = new DynamoDBClient({
  maxAttempts: 5,
});
```

También puede controlar el tiempo de los reintentos con una estrategia de reintentos personalizada. Para ello, importe el paquete de utilidades `util-retry` y cree una función de espera personalizada que calcule el tiempo de espera entre reintentos en función del recuento de reintentos actual.

En el siguiente ejemplo, se recomienda realizar un máximo de 5 intentos con retardos de 15, 30, 90 y 360 milisegundos en caso de que falle el primer intento. La función de espera personalizada `calculateRetryBackoff` calcula los retardos aceptando el número de reintentos (comienza con 1 en el primer intento) y devuelve el número de milisegundos que hay que esperar para dicha solicitud.

```
const { ConfiguredRetryStrategy } = require("@aws-sdk/util-retry");

const calculateRetryBackoff = (attempt) => {
  const backoffTimes = [15, 30, 90, 360];
  return backoffTimes[attempt - 1] || 0;
};

const client = new DynamoDBClient({
  retryStrategy: new ConfiguredRetryStrategy(
    5, // max attempts.
    calculateRetryBackoff // backoff function.
  ),
});
```

Esperadores

El cliente DynamoDB incluye dos funciones de esperador muy útiles que se pueden utilizar al crear, modificar o eliminar tablas si desea que el código espere hasta que finalice la modificación de la tabla. Por ejemplo, puede desplegar una tabla y llamar a la función `waitUntilTableExists`. El código se bloqueará hasta que la tabla se ACTIVE. El esperador sondea internamente el servicio DynamoDB con una `describe-table` cada 20 segundos.

```
import {waitUntilTableExists, waitUntilTableNotExists} from "@aws-sdk/client-dynamodb";

… <create table details>

const results = await waitUntilTableExists({client: client, maxWaitTime: 180},
 {TableName: "products"});
if (results.state == 'SUCCESS') {
  return results.reason.Table
}
console.error(`${results.state} ${results.reason}`);
```

La característica `waitUntilTableExists` devuelve el control solo cuando puede ejecutar un comando `describe-table` que muestre el estado ACTIVE de la tabla. Esto garantiza que podrá usar `waitUntilTableExists` para esperar a que se complete la creación, así como realizar modificaciones, como añadir un índice GSI, que puede tardar algún tiempo en aplicarse antes de que la tabla vuelva al estado ACTIVE.

Gestión de errores

En los primeros ejemplos, hemos detectado todos los errores ampliamente. Sin embargo, en las aplicaciones prácticas, es importante discernir entre varios tipos de error e implementar una gestión de errores más precisa.

Las respuestas de error de DynamoDB contienen metadatos, incluido el nombre del error. Puede detectar los errores y luego compararlos con los posibles nombres de cadena de las condiciones de error para determinar cómo continuar. En el caso de los errores del servidor, puede utilizar el operador `instanceof` con los tipos de error exportados por el paquete `@aws-sdk/client-dynamodb` para administrar eficazmente la gestión de errores.

Es importante tener en cuenta que estos errores solo se manifiestan una vez agotados todos los reintentos. Si se vuelve a intentar un error y, finalmente, se realiza una llamada correcta, desde el punto de vista del código, no se trata de ningún error, sino de una latencia ligeramente elevada. Los reintentos se mostrarán en los gráficos de Amazon CloudWatch como solicitudes fallidas, como las solicitudes de limitación o de error. Si el cliente alcanza el número máximo de reintentos, se dará por vencido y generará una excepción. Esta es la forma que tiene el cliente de decir que no va a volver a intentarlo.

A continuación se muestra un fragmento para detectar el error y tomar medidas en función del tipo de error devuelto.

```
import {
  ResourceNotFoundException
  ProvisionedThroughputExceededException,
  DynamoDBServiceException,
} from "@aws-sdk/client-dynamodb";

try {
  await client.send(someCommand);
} catch (e) {
    if (e instanceof ResourceNotFoundException) {
      // Handle ResourceNotFoundException
    } else if (e instanceof ProvisionedThroughputExceededException) {
```

```
      // Handle ProvisionedThroughputExceededException
    } else if (e instanceof DynamoDBServiceException) {
      // Handle DynamoDBServiceException
    } else {
      // Other errors such as those from the SDK
      if (e.name === "TimeoutError") {
        // Handle SDK TimeoutError.
      } else {
        // Handle other errors.
      }
    }
}
```

Consulte the section called “Gestión de errores” para ver las cadenas de error más comunes en la Guía para desarrolladores de DynamoDB. Los errores exactos que se pueden producir con cualquier llamada a la API en concreto aparecen en la documentación de esa llamada a la API, como los documentos de la API Query.

Los metadatos de los errores incluyen propiedades adicionales, en función del error. En el caso de un `TimeoutError`, los metadatos incluyen el número de intentos realizados y el `totalRetryDelay`, tal y como se muestra a continuación.

```
{
  "name": "TimeoutError",
  "$metadata": {
    "attempts": 3,
    "totalRetryDelay": 199
  }
}
```

Si administra su propia política de reintentos, se recomienda diferenciar entre limitaciones y errores:

- Una limitación (indicada con una `ProvisionedThroughputExceededException` o `ThrottlingException`) representa un servicio en buen estado que le informa de que ha superado la capacidad de lectura o escritura en una tabla o partición de DynamoDB. Cada milisegundo que pasa, se dispone de un poco más de capacidad de lectura o escritura, por lo que puede volver a intentarlo rápidamente (por ejemplo, cada 50 ms) para intentar acceder a la capacidad recién liberada.

  Con las limitaciones no se requiere un retroceso exponencial, pues las limitaciones son ligeras para que DynamoDB las devuelva. Además, no se le cobra por solicitud. El retroceso exponencial

asigna retardos más prolongados a los subprocesos de los clientes que ya han esperado más tiempo, lo que, estadísticamente, prolonga la p50 y la p99 hacia fuera.

- Un error (indicado por un `InternalServerError` o un `ServiceUnavailable`, entre otros) representa un problema transitorio con el servicio, posiblemente de toda la tabla o solo de la partición desde la que está leyendo o escribiendo. En el caso de los errores, puede hacer una pausa más larga antes de volver a intentarlo, por ejemplo, 250 ms o 500 ms, y utilizar la fluctuación para escalonar los reintentos.

Registro

Active el registro para obtener más información sobre lo que hace el SDK. Puede establecer un parámetro en el `DynamoDBClient` tal como se muestra en el ejemplo siguiente. Aparecerá más información de registro en la consola que incluirá metadatos como el código de estado y la capacidad consumida. Si ejecuta el código de forma local en una ventana de terminal, los registros aparecen allí. Si ejecuta el código en AWS Lambda y tiene configurados los registros de Amazon CloudWatch, el resultado de la consola se escribirá allí.

```
const client = new DynamoDBClient({
  logger: console
});
```

También puede enlazar con las actividades internas del SDK y realizar un registro personalizado a medida que se produzcan determinados eventos. En el siguiente ejemplo se utiliza la `middlewareStack` del cliente para interceptar cada solicitud a medida que se envía desde el SDK y la registra a medida que se produce.

```
const client = new DynamoDBClient({});

client.middlewareStack.add(
  (next) => async (args) => {
    console.log("Sending request from AWS SDK", { request: args.request });
    return next(args);
  },
  {
    step: "build",
    name: "log-ddb-calls",
  }
);
```

La `MiddlewareStack` es un enlace potente para observar y controlar el comportamiento del SDK. Consulte el blog Introducing Middleware Stack in Modular AWS SDK para JavaScript para obtener más información.

Consideraciones

A la hora de implementar AWS SDK para JavaScript en su proyecto, debe considerar algunos factores adicionales.

Sistemas de módulos

El SDK admite dos sistemas de módulos: CommonJS y ES (ECMAScript). CommonJS usa la función `require`, mientras que ES usa la palabra clave `import`.

1. Common JS: `const { DynamoDBClient, PutItemCommand } = require("@aws-sdk/client-dynamodb");`
2. ES (ECMAScript: `import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";`

El tipo de proyecto dicta el sistema de módulos que se utilizará y se especifica en la sección de tipos del archivo package.json. El valor predeterminado es CommonJS. Use `"type": "module"` para indicar un proyecto ES. Si ya tiene un proyecto Node.JS que utiliza el formato de paquete CommonJS, puede añadir funciones con la sintaxis de importación del SDK V3 más moderna, si asigna la extensión .mjs a los archivos de funciones. Esto permitirá que el archivo de código se trate como ES (ECMAScript).

Operaciones asíncronas

Verá muchos ejemplos de código que utilizan callbacks y que prometen gestionar el resultado de las operaciones de DynamoDB. Con el JavaScript moderno, esta complejidad ya no es necesaria y los desarrolladores pueden aprovechar la sintaxis async/await, más sucinta y legible, para las operaciones asíncronas.

Tiempo de ejecución del navegador web

Los desarrolladores web y móviles que creen con React o React Native pueden usar el SDK para JavaScript en sus proyectos. Con la V2 anterior del SDK, los desarrolladores web tenían que cargar el SDK completo en el navegador y hacer referencia a una imagen del SDK alojada en https://sdk.amazonaws.com/js/.

Con la V3, se pueden agrupar solo los módulos del cliente V3 necesarios y todas las funciones de JavaScript requeridas en un solo archivo JavaScript mediante Webpack. Este archivo se puede

añadir en una etiqueta de script en el <head> de sus páginas HTML, tal como se explica en la sección Comenzar en el navegador de la documentación del SDK.

Operaciones del plano de datos de DAX

Las operaciones del plano de datos de Amazon DynamoDB Streams Accelerator (DAX) son compatibles con el SDK para JavaScript V3.

Programación de DynamoDB con AWS SDK for Java 2.x

Esta guía de programación ofrece orientación a los programadores que desean utilizar Amazon DynamoDB con Java. La guía abarca diferentes conceptos, como las capas de abstracción, la administración de la configuración, la gestión de errores, el control de las políticas de reintentos y la administración de keep-alive.

Temas

- Acerca del AWS SDK for Java 2.x
- Introducción a la AWS SDK for Java 2.x
- Revisión de la documentación de AWS SDK for Java 2.x
- Interfaces admitidas
- Ejemplos de código adicionales
- Programación síncrona y asíncrona
- Clientes de HTTP
- Configuración de un cliente HTTP
- Gestión de errores
- ID de solicitud de AWS
- Registro
- Paginación
- Anotaciones de clases de datos

Acerca del AWS SDK for Java 2.x

Puede acceder a DynamoDB desde Java utilizando el AWS SDK para Java oficial. El SDK para Java tiene dos versiones: 1.x y 2.x. El 12 de enero de 2024 se anunció el fin de soporte de la versión 1.x. El 31 de julio de 2024 entrará en modo de mantenimiento y está previsto que se deje de dar soporte

el 31 de diciembre de 2025. Para nuevos desarrollos, recomendamos encarecidamente el uso de la versión 2.x, que se lanzó por primera vez en 2018. Esta guía hace referencia exclusivamente a la versión 2.x y se centra únicamente en las partes del SDK relevantes para DynamoDB.

Para obtener más información sobre el mantenimiento y el soporte de los AWS SDK, consulte AWS SDK and Tools maintenance policy y AWS SDKs and Tools version support matrix en la Guía de referencia de los SDK y las herramientas de AWS.

AWS SDK for Java 2.x constituye un cambio importante con respecto a la base de código de la versión 1.x. El SDK para Java 2.x admite las características modernas de Java, como la E/S sin bloqueo introducida en Java 8. El SDK para Java 2.x también admite implementaciones de clientes HTTP conectables para ofrecer más flexibilidad y opciones de configuración para las conexiones de red.

Un cambio notable entre el SDK para Java 1.x y el SDK para Java 2.x es el uso de un nuevo nombre del paquete. El SDK para Java 1.x usa el nombre del paquete `com.amazonaws`, mientras que el SDK para Java 2.x usa `software.amazon.awssdk`. Del mismo modo, los artefactos de Maven para el SDK para Java 1.x utilizan el `groupId` de `com.amazonaws`, mientras que los artefactos del SDK para Java 2.x utilizan el `groupId` de `software.amazon.awssdk`.

> **Important**
>
> El AWS SDK para Java 1.x tiene un paquete de DynamoDB denominado `com.amazonaws.dynamodbv2`. El "v2" en el nombre del paquete no indica que sea para Java 2 (J2SE). Indica que el paquete admite la segunda versión de la API de bajo nivel de DynamoDB en lugar de la versión original de la API de bajo nivel.

Compatibilidad con las versiones de Java

El AWS SDK for Java 2.x proporciona compatibilidad total con las versiones de Java basadas en soporte a largo plazo (LTS).

Introducción a la AWS SDK for Java 2.x

En el siguiente tutorial se muestra cómo utilizar Apache Maven para definir las dependencias del SDK para Java 2.x. En este tutorial también se explica cómo escribir el código que se conecta a DynamoDB para enumerar las tablas de DynamoDB disponibles. El tutorial de esta guía se basa en el tutorial Get started with the AWS SDK for Java 2.x de la Guía para desarrolladores de AWS

SDK for Java 2.x. Hemos modificado este tutorial para realizar llamadas a DynamoDB en lugar de a Amazon S3.

Tendrá que hacer lo siguiente para completar este tutorial:

- Paso 1: Prepararse para el tutorial
- Paso 2: Crear el proyecto
- Paso 3: Escribir el código
- Paso 4: Compilar y ejecutar la aplicación

Paso 1: Prepararse para el tutorial

Antes de empezar este tutorial, necesitará:

- Permiso para acceder a DynamoDB.
- Un entorno de desarrollo de Java que esté configurado con acceso de inicio de sesión único a los Servicios de AWS para utilizar el Portal de acceso a AWS.

Para prepararse para este tutorial, siga las instrucciones de Setup overview de la Guía para desarrolladores de AWS SDK for Java 2.x. Cuando haya configurado su entorno de desarrollo con acceso de inicio de sesión único para el SDK de Java y tenga una sesión activa en el portal de acceso a AWS, continúe con el paso 2 de este tutorial.

Paso 2: Crear el proyecto

Para crear el proyecto de este tutorial, ejecute un comando de Maven que le pida información sobre cómo configurar el proyecto. Una vez introducidas y confirmadas todas las entradas, Maven finaliza la construcción del proyecto al crear un archivo `pom.xml` y archivos stub de Java.

1. Abra una ventana de terminal o línea de comandos y acceda a un directorio de su elección, como su carpeta `Desktop` o `Home`.
2. Introduzca el siguiente comando en el terminal y pulse Intro.

```
mvn archetype:generate \
   -DarchetypeGroupId=software.amazon.awssdk \
   -DarchetypeArtifactId=archetype-app-quickstart \
   -DarchetypeVersion=2.22.0
```

3. Cuando se le solicite, introduzca el valor que aparece en la segunda columna.

Prompt	Valor para introducir
`Define value for property 'service':`	`dynamodb`
`Define value for property 'httpClient' :`	`apache-client`
`Define value for property 'nativeImage' :`	`false`
`Define value for property 'credentialProvider'`	`identity-center`
`Define value for property 'groupId':`	`org.example`
`Define value for property 'artifactId':`	`getstarted`
`Define value for property 'version' 1.0-SNAPSHOT:`	*<Enter>*
`Define value for property 'package' org.example:`	*<Enter>*

4. Después de introducir el último valor, Maven muestra una lista de las opciones que ha elegido. Para confirmarlas, introduzca Y. O bien, introduzca N y, a continuación, vuelva a introducir las opciones.

Maven crea una carpeta del proyecto llamada `getstarted` basándose en el valor de `artifactId` que haya introducido. En la carpeta `getstarted`, busque un archivo llamado `README.md` que pueda revisar, un archivo `pom.xml` y un directorio `src`.

Maven crea el árbol de directorios siguiente.

```
getstarted
 ### README.md
 ### pom.xml
```

```
 ### src
     ### main
     #   ### java
     #   #   ### org
     #   #       ### example
     #   #           ### App.java
     #   #           ### DependencyFactory.java
     #   #           ### Handler.java
     #   ### resources
     #       ### simplelogger.properties
     ### test
         ### java
             ### org
                 ### example
                     ### HandlerTest.java

 10 directories, 7 files
```

A continuación se muestra el contenido del archivo de proyecto `pom.xml`.

`pom.xml`

La sección `dependencyManagement` contiene una dependencia a AWS SDK for Java 2.x y la sección `dependencies` a Amazon DynamoDB. La especificación de estas dependencias obliga a Maven a incluir los archivos `.jar` correspondientes en la ruta de clases de Java. De forma predeterminada, el SDK de AWS no incluye todas las clases para todos los Servicios de AWS. En el caso de DynamoDB, si utiliza la interfaz de bajo nivel, debería tener una dependencia en el artefacto `dynamodb`. O bien, si utiliza la interfaz de alto nivel, en el artefacto `dynamodb-enhanced`. Si no incluye las dependencias pertinentes, su código no puede compilarse. El proyecto usa Java 1.8 debido al valor `1.8` de las `maven.compiler.target` propiedades `maven.compiler.source`.

```
<?xml version="1.0" encoding="UTF-8"?>
 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
     <modelVersion>4.0.0</modelVersion>

     <groupId>org.example</groupId>
     <artifactId>getstarted</artifactId>
     <version>1.0-SNAPSHOT</version>
     <packaging>jar</packaging>
     <properties>
```

```
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <maven.shade.plugin.version>3.2.1</maven.shade.plugin.version>
        <maven.compiler.plugin.version>3.6.1</maven.compiler.plugin.version>
        <exec-maven-plugin.version>1.6.0</exec-maven-plugin.version>
        <aws.java.sdk.version>2.22.0</aws.java.sdk.version> <-------- SDK version
picked up from archetype version.
        <slf4j.version>1.7.28</slf4j.version>
        <junit5.version>5.8.1</junit5.version>
    </properties>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>software.amazon.awssdk</groupId>
                <artifactId>bom</artifactId>
                <version>${aws.java.sdk.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>dynamodb</artifactId>  <-------- DynamoDB dependency
            <exclusions>
                <exclusion>
                    <groupId>software.amazon.awssdk</groupId>
                    <artifactId>netty-nio-client</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>software.amazon.awssdk</groupId>
                    <artifactId>apache-client</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>sso</artifactId> <-------- Required for identity center
authentication.
```

```
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>ssooidc</artifactId> <-------- Required for identity center
authentication.
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>apache-client</artifactId> <-------- HTTP client specified.
            <exclusions>
                <exclusion>
                    <groupId>commons-logging</groupId>
                    <artifactId>commons-logging</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <!-- Needed to adapt Apache Commons Logging used by Apache HTTP Client to
Slf4j to avoid
        ClassNotFoundException: org.apache.commons.logging.impl.LogFactoryImpl during
runtime -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>jcl-over-slf4j</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <!-- Test Dependencies -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
```

```
            <artifactId>junit-jupiter</artifactId>
            <version>${junit5.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>${maven.compiler.plugin.version}</version>
            </plugin>
        </plugins>
    </build>

</project>
```

Paso 3: Escribir el código

En el código siguiente se muestra la clase App que crea Maven. El método `main` es el punto de entrada a la aplicación, que crea una instancia de la `Handler` clase y, a continuación, llama a su método `sendRequest`.

Clase **App**

```
package org.example;
 import org.slf4j.Logger;
 import org.slf4j.LoggerFactory;

 public class App {
     private static final Logger logger = LoggerFactory.getLogger(App.class);

     public static void main(String... args) {
         logger.info("Application starts");

         Handler handler = new Handler();
         handler.sendRequest();

         logger.info("Application ends");
     }
 }
```

La clase `DependencyFactory` que crea Maven contiene el método de fábrica `dynamoDbClient` que crea y devuelve una instancia de `DynamoDbClient`. La instancia `DynamoDbClient` usa una instancia del cliente HTTP basado en Apache. Esto se debe a que especificó `apache-client` cuando Maven preguntó qué cliente HTTP usar.

En el siguiente código se muestra la clase `DependencyFactory`.

Clase DependencyFactory

```
package org.example;

 import software.amazon.awssdk.http.apache.ApacheHttpClient;
 import software.amazon.awssdk.services.dynamodb.DynamoDbClient;

 /**
  * The module containing all dependencies required by the {@link Handler}.
  */
 public class DependencyFactory {

     private DependencyFactory() {}

     /**
      * @return an instance of DynamoDbClient
      */
     public static DynamoDbClient dynamoDbClient() {
         return DynamoDbClient.builder()
                         .httpClientBuilder(ApacheHttpClient.builder())
                         .build();
     }
 }
```

La clase `Handler` contiene la lógica principal del programa. Cuando se crea una instancia de `Handler` en la clase App, `DependencyFactory` proporciona el cliente de servicios `DynamoDbClient`. El código utiliza la instancia de `DynamoDbClient` para llamar a DynamoDB.

Maven genera la siguiente clase `Handler` con un comentario *`TODO`*. El siguiente paso del tutorial reemplaza el comentario *`TODO`* por código.

Clase **`Handler`**, generada por Maven

```
package org.example;
```

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;


public class Handler {
    private final DynamoDbClient dynamoDbClient;

    public Handler() {
        dynamoDbClient = DependencyFactory.dynamoDbClient();
    }

    public void sendRequest() {
        // TODO: invoking the API calls using dynamoDbClient.
    }
}
```

Para completar la lógica, reemplace todo el contenido de la clase `Handler` por el código siguiente. El método `sendRequest` se rellena y se añaden las importaciones necesarias.

Clase **`Handler`**, implementada

El siguiente código usa la instancia `DynamoDbClient` para recuperar una lista de tablas existentes. Si existen tablas para una cuenta y una Región de AWS determinadas, el código usa la instancia de `Logger` para registrar los nombres de esas tablas.

```
package org.example;

 import org.slf4j.Logger;
 import org.slf4j.LoggerFactory;
 import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
 import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;


 public class Handler {
     private final DynamoDbClient dynamoDbClient;

     public Handler() {
         dynamoDbClient = DependencyFactory.dynamoDbClient();
     }

     public void sendRequest() {
         Logger logger = LoggerFactory.getLogger(Handler.class);
```

```
        logger.info("calling the DynamoDB API to get a list of existing tables");
        ListTablesResponse response = dynamoDbClient.listTables();

        if (!response.hasTableNames()) {
            logger.info("No existing tables found for the configured account &
region");
        } else {
            response.tableNames().forEach(tableName -> logger.info("Table: " +
tableName));
        }
    }
}
```

Paso 4: Compilar y ejecutar la aplicación

Una vez creado el proyecto y que contenga la clase `Handler` completa, compile y ejecute la aplicación.

1. Asegúrese de que tenga una sesión de AWS IAM Identity Center activa. Para ello, ejecute el comando de la AWS Command Line Interface (AWS CLI) `aws sts get-caller-identity` y compruebe la respuesta. Si no tiene una sesión activa, consulte Sign in using the AWS CLI para obtener instrucciones.
2. Abra una ventana de terminal o símbolo del sistema y desplácese hasta el directorio del proyecto `getstarted`.
3. Para compilar su proyecto, ejecute el comando siguiente:

   ```
   mvn clean package
   ```

4. Para ejecutar la aplicación, ejecute el comando siguiente:

   ```
   mvn exec:java -Dexec.mainClass="org.example.App"
   ```

Después de ver el archivo, elimine el objeto y, a continuación, elimine el bucket.

Success

Si su proyecto de Maven se creó y ejecutó sin errores, ¡enhorabuena! Ha creado correctamente su primera aplicación de Java con el SDK para Java 2.x.

Eliminación

Para limpiar los recursos que ha creado en este tutorial, elimine la carpeta del proyecto `getstarted`.

Revisión de la documentación de AWS SDK for Java 2.x

La Guía para desarrolladores de AWS SDK for Java 2.x cubre todos los aspectos del SDK en todos los Servicios de AWS. Le aconsejamos que revise primero los siguientes temas:

- Migrate from version 1.x to 2.x: incluye una explicación detallada de las diferencias entre la versión 1.x y la 2.x. Este tema también incluye instrucciones sobre cómo utilizar ambas versiones principales en paralelo.
- DynamoDB guide for Java 2.x SDK: muestra cómo realizar operaciones básicas de DynamoDB, tales como crear una tabla, manipular elementos y recuperar elementos. En estos ejemplos se utiliza la interfaz de bajo nivel. Java tiene varias interfaces, tal como se explica en la siguiente sección: Interfaces admitidas.

> **Tip**
>
> Tras revisar estos temas, agregue a sus favoritos la Referencia de la API de AWS SDK for Java 2.x. Abarca todos los Servicios de AWS y le recomendamos que la utilice como referencia principal de la API.

Interfaces admitidas

AWS SDK for Java 2.x admite las siguientes interfaces en función del nivel de abstracción que desee.

Temas de esta sección

- Interfaz de bajo nivel
- Interfaz de alto nivel
- Interfaz de documentos
- Comparación de interfaces con un ejemplo de Query

Interfaz de bajo nivel

La interfaz de bajo nivel proporciona una asignación uno a uno con la API del servicio subyacente. Todas las API de DynamoDB están disponibles a través de esta interfaz. Esto significa que la interfaz de bajo nivel puede proporcionar una funcionalidad completa, pero suele ser más detallada y su uso es más complejo. Por ejemplo, debe usar las funciones `.s()` para incluir cadenas y las funciones `.n()` para incluir números. En el siguiente ejemplo de PutItem se inserta un elemento mediante la interfaz de bajo nivel.

```
import org.slf4j.*;
import software.amazon.awssdk.http.crt.AwsCrtHttpClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.*;

import java.util.Map;

public class PutItem {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbClient DYNAMODB_CLIENT = DynamoDbClient.create();
    private static final Logger LOGGER = LoggerFactory.getLogger(PutItem.class);

    private void putItem() {
        PutItemResponse response = DYNAMODB_CLIENT.putItem(PutItemRequest.builder()
                .item(Map.of(
                        "pk", AttributeValue.builder().s("123").build(),
                        "sk", AttributeValue.builder().s("cart#123").build(),
                        "item_data",
 AttributeValue.builder().s("YourItemData").build(),
                        "inventory", AttributeValue.builder().n("500").build()
                        // ... more attributes ...
                ))
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .tableName("YourTableName")
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }
}
```

Interfaz de alto nivel

La interfaz de alto nivel de AWS SDK for Java 2.x se denomina cliente mejorado de DynamoDB. Esta interfaz proporciona una experiencia de creación de código más idiomática.

El cliente mejorado ofrece una forma de asignar entre las clases de datos del lado del cliente y las tablas de DynamoDB diseñadas para almacenar esos datos. Puede definir las relaciones entre las tablas y sus correspondientes clases de modelo en el código. Luego, puede confiar en el SDK para administrar la manipulación de los tipos de datos. Para obtener más información sobre el cliente mejorado, consulte DynamoDB enhanced client API en la Guía parea desarrolladores de AWS SDK for Java 2.x.

El siguiente ejemplo de PutItem utiliza la interfaz de alto nivel. En este ejemplo, el `DynamoDbBean` denominado `YourItem` crea un `TableSchema` que permite su uso directo como entrada para la llamada `putItem()`.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientPutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourItem> DYNAMODB_TABLE =
 ENHANCED_DYNAMODB_CLIENT.table("YourTableName", TableSchema.fromBean(YourItem.class));
    private static final Logger LOGGER = LoggerFactory.getLogger(PutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<YourItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourItem.class)
                .item(new YourItem("123", "cart#123", "YourItemData", 500))
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }

    @DynamoDbBean
    public static class YourItem {
```

```
        public YourItem() {}

        public YourItem(String pk, String sk, String itemData, int inventory) {
            this.pk = pk;
            this.sk = sk;
            this.itemData = itemData;
            this.inventory = inventory;
        }

        private String pk;
        private String sk;
        private String itemData;

        private int inventory;

        @DynamoDbPartitionKey
        public void setPk(String pk) {
            this.pk = pk;
        }

        public String getPk() {
            return pk;
        }

        @DynamoDbSortKey
        public void setSk(String sk) {
            this.sk = sk;
        }

        public String getSk() {
            return sk;
        }

        public void setItemData(String itemData) {
            this.itemData = itemData;
        }

        public String getItemData() {
            return itemData;
        }

        public void setInventory(int inventory) {
            this.inventory = inventory;
        }
```

```
        public int getInventory() {
            return inventory;
        }
    }
}
```

AWS SDK para Java 1.x tiene su propia interfaz de alto nivel, a la que suele hacer referencia su `DynamoDBMapper` de clase principal. El AWS SDK for Java 2.x se publica en un paquete separado (y un artefacto de Maven) denominado `software.amazon.awssdk.enhanced.dynamodb`. El SDK para Java 2.x a menudo se denomina por su `DynamoDbEnhancedClient` de clase principal.

Interfaz de alto nivel que utiliza clases de datos inmutables

La característica de asignación de la API de cliente mejorado de DynamoDB también funciona con clases de datos inmutables. Una clase inmutable solo tiene getters y requiere una clase constructora que el SDK utiliza para crear instancias de la clase. La inmutabilidad en Java es un estilo común que los desarrolladores pueden usar para crear clases sin efectos secundarios. El comportamiento de estas clases es más predecible en aplicaciones complejas de varios subprocesos. En lugar de usar la anotación `@DynamoDbBean` como se muestra en el High-level interface example, las clases inmutables usan la anotación `@DynamoDbImmutable`, que toma la clase de generador como entrada.

En el siguiente ejemplo, se utiliza la clase de generador `DynamoDbEnhancedClientImmutablePutItem` como entrada para crear un esquema de tabla. A continuación, el ejemplo proporciona el esquema como entrada para la llamada a la API PutItem.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientImmutablePutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourImmutableItem>
 DYNAMODB_TABLE = ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromImmutableClass(YourImmutableItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientImmutablePutItem.class);
```

```
    private void putItem() {
        PutItemEnhancedResponse<YourImmutableItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourImmutableItem.class)
                .item(YourImmutableItem.builder()
                                    .pk("123")
                                    .sk("cart#123")
                                    .itemData("YourItemData")
                                    .inventory(500)
                                    .build())
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }
}
```

En el siguiente ejemplo, se muestra la clase de datos inmutable.

```
@DynamoDbImmutable(builder = YourImmutableItem.YourImmutableItemBuilder.class)
class YourImmutableItem {
    private final String pk;
    private final String sk;
    private final String itemData;
    private final int inventory;
    public YourImmutableItem(YourImmutableItemBuilder builder) {
        this.pk = builder.pk;
        this.sk = builder.sk;
        this.itemData = builder.itemData;
        this.inventory = builder.inventory;
    }

    public static YourImmutableItemBuilder builder() { return new
 YourImmutableItemBuilder(); }

    @DynamoDbPartitionKey
    public String getPk() {
        return pk;
    }

    @DynamoDbSortKey
    public String getSk() {
        return sk;
    }
```

```
    public String getItemData() {
        return itemData;
    }

    public int getInventory() {
        return inventory;
    }

    static final class YourImmutableItemBuilder {
        private String pk;
        private String sk;
        private String itemData;
        private int inventory;

        private YourImmutableItemBuilder() {}

        public YourImmutableItemBuilder pk(String pk) { this.pk = pk; return this; }
        public YourImmutableItemBuilder sk(String sk) { this.sk = sk; return this; }
        public YourImmutableItemBuilder itemData(String itemData) { this.itemData =
 itemData; return this; }
        public YourImmutableItemBuilder inventory(int inventory) { this.inventory =
 inventory; return this; }

        public YourImmutableItem build() { return new YourImmutableItem(this); }
    }
}
```

Interfaz de alto nivel que utiliza clases de datos inmutables y bibliotecas de generación reutilizables de terceros

Las clases de datos inmutables (que se muestran en el ejemplo anterior) requieren algo de código reutilizable. Por ejemplo, la lógica de getter y setter en las clases de datos, además de las clases `Builder`. Las bibliotecas de terceros, como Project Lombok, pueden ayudarlo a generar ese tipo de código reutilizable. Reducir la mayor parte del código reutilizable ayuda a limitar la cantidad de código necesaria para trabajar con clases de datos inmutables y con el SDK de AWS. Además, esto mejora la productividad y la legibilidad del código. Para obtener más información, consulte Use third-party libraries, such as Lombok en la Guía para desarrolladores de AWS SDK for Java 2.x.

En el siguiente ejemplo se muestra cómo Project Lombok simplifica el código necesario para utilizar la API de cliente mejorada de DynamoDB.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedClientImmutableLombokPutItem {

    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourImmutableLombokItem>
 DYNAMODB_TABLE = ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromImmutableClass(YourImmutableLombokItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientImmutableLombokPutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<YourImmutableLombokItem> response =
 DYNAMODB_TABLE.putItemWithResponse(PutItemEnhancedRequest.builder(YourImmutableLombokItem.clas
                .item(YourImmutableLombokItem.builder()
                        .pk("123")
                        .sk("cart#123")
                        .itemData("YourItemData")
                        .inventory(500)
                        .build())
                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
    }
}
```

En el siguiente ejemplo se muestra el objeto de datos inmutables de la clase de datos inmutables.

```
import lombok.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;

@Builder
@DynamoDbImmutable(builder =
 YourImmutableLombokItem.YourImmutableLombokItemBuilder.class)
@Value
public class YourImmutableLombokItem {

    @Getter(onMethod_=@DynamoDbPartitionKey)
```

```
    String pk;
    @Getter(onMethod_=@DynamoDbSortKey)
    String sk;
    String itemData;
    int inventory;
}
```

La clase `YourImmutableLombokItem` usa las siguientes anotaciones proporcionadas por Project Lombok y el AWS SDK:

- @Builder: produce API de creador complejas para las clases de datos que proporciona Project Lombok.
- @DynamoDbImmutable: identifica la clase `DynamoDbImmutable` como una anotación de entidad asignable de DynamoDB proporcionada por el AWS SDK.
- @Value: la variante inmutable de `@Data`. Todos los campos se convierten en privados y definitivos de forma predeterminada y los setters no se generan. El proyecto Lombok proporciona esta anotación.

Interfaz de documentos

La interfaz de documentos de AWS SDK for Java 2.x evita la necesidad de especificar descriptores de tipos de datos. Los tipos de datos quedan implícitos en la propia semántica de los datos. Esta interfaz de documentos es similar a la interfaz de documentos de AWS SDK para Java 1.x, pero se ha rediseñado.

El siguiente Document interface example muestra la llamada `PutItem` expresada mediante la interfaz de documentos. En el ejemplo también se utiliza EnhancedDocument. Para ejecutar comandos en una tabla de DynamoDB mediante la API de documentos mejorada, primero debe asociar la tabla al esquema de la tabla de documentos para crear un objeto de recurso `DynamoDBTable`. El generador de un esquema de tabla de documentos requiere la clave de índice principal y proveedores de convertidores de atributos.

Puede utilizar `AttributeConverterProvider.defaultProvider()` para convertir los atributos de los documentos de los tipos predeterminados. Puede cambiar el comportamiento predeterminado general con una implementación de `AttributeConverterProvider` personalizada. También puede cambiar el conversor de un solo atributo. En la Guía de referencia de SDK y herramientas de AWS se proporcionan más detalles y ejemplos sobre cómo utilizar los convertidores personalizados. Su uso principal es para los atributos de las clases de dominio que no tienen un conversor

predeterminado disponible. Con un conversor personalizado, puede proporcionar al SDK la información necesaria para escribir o leer en DynamoDB.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.document.EnhancedDocument;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.ReturnConsumedCapacity;

public class DynamoDbEnhancedDocumentClientPutItem {
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<EnhancedDocument> DYNAMODB_TABLE =
            ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.documentSchemaBuilder()

 .addIndexPartitionKey(TableMetadata.primaryIndexName(),"pk", AttributeValueType.S)
                            .addIndexSortKey(TableMetadata.primaryIndexName(), "sk",
 AttributeValueType.S)

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                            .build());

    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedDocumentClientPutItem.class);

    private void putItem() {
        PutItemEnhancedResponse<EnhancedDocument> response =
 DYNAMODB_TABLE.putItemWithResponse(
                        PutItemEnhancedRequest.builder(EnhancedDocument.class)
                                .item(
                                    EnhancedDocument.builder()

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                                            .putString("pk", "123")
                                            .putString("sk", "cart#123")
                                            .putString("item_data", "YourItemData")
                                            .putNumber("inventory", 500)
                                            .build())
                                .returnConsumedCapacity(ReturnConsumedCapacity.TOTAL)
                                .build());
        LOGGER.info("PutItem call consumed [" +
 response.consumedCapacity().capacityUnits() + "] Write Capacity Unites (WCU)");
```

```
    }

}
```

Para convertir documentos JSON a los tipos de datos nativos de Amazon DynamoDB y viceversa, puede utilizar los siguientes métodos de la utilidad:

- `EnhancedDocument.fromJson(String json)`: crea una nueva instancia de EnhancedDocument a partir de una cadena JSON.
- `EnhancedDocument.toJson()`: crea una representación de la cadena JSON del documento que puede utilizar en su aplicación como cualquier otro objeto JSON.

Comparación de interfaces con un ejemplo de **`Query`**

En esta sección, se muestra la misma llamada a `Query` expresada mediante las distintas interfaces. Para ajustar los resultados de estas consultas, tenga en cuenta lo siguiente:

- DynamoDB se centra en un valor de clave de partición concreto, por lo que debe especificar la clave de partición por completo.
- Para que la consulta se dirija únicamente a los artículos del carrito, la clave de clasificación tiene una expresión de condición clave que utiliza `begins_with`.
- Utilizamos `limit()` para que la consulta devuelva un máximo de 100 elementos.
- Establecemos el valor `scanIndexForward` en false. Los resultados se muestran en orden de bytes en UTF-8, lo que normalmente significa que el artículo del carrito con el número más bajo se devuelve primero. Al establecer `scanIndexForward` en false, invertimos el orden y se devuelve primero el elemento del carrito con el número más alto.
- Aplicamos un filtro para eliminar cualquier resultado que no coincida con los criterios. Los datos que se filtran consumen capacidad de lectura independientemente de si el elemento coincide con el filtro.

Example **`Query`** mediante la interfaz de bajo nivel

En el siguiente ejemplo, se consulta una tabla denominada `YourTableName` mediante una `keyConditionExpression`. Esto limita la consulta a un valor de clave de partición específico y a un valor de clave de clasificación que comience por un valor de prefijo específico. Estas condiciones clave limitan la cantidad de datos leídos de DynamoDB. Por último, la consulta aplica un filtro a los datos obtenidos de DynamoDB mediante una `filterExpression`.

```
import org.slf4j.*;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.*;

import java.util.Map;

public class Query {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbClient DYNAMODB_CLIENT =
 DynamoDbClient.builder().build();
    private static final Logger LOGGER = LoggerFactory.getLogger(Query.class);

    private static void query() {
        QueryResponse response = DYNAMODB_CLIENT.query(QueryRequest.builder()
                .expressionAttributeNames(Map.of("#name", "name"))
                .expressionAttributeValues(Map.of(
                    ":pk_val", AttributeValue.fromS("id#1"),
                    ":sk_val", AttributeValue.fromS("cart#"),
                    ":name_val", AttributeValue.fromS("SomeName")))
                .filterExpression("#name = :name_val")
                .keyConditionExpression("pk = :pk_val AND begins_with(sk, :sk_val)")
                .limit(100)
                .scanIndexForward(false)
                .tableName("YourTableName")
                .build());

        LOGGER.info("nr of items: " + response.count());
        LOGGER.info("First item pk: " + response.items().get(0).get("pk"));
        LOGGER.info("First item sk: " + response.items().get(0).get("sk"));
    }
}
```

Example **`Query`** mediante la interfaz de documentos

En el siguiente ejemplo, se consulta una tabla denominada `YourTableName` mediante la interfaz de documentos.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.document.EnhancedDocument;
```

```
import software.amazon.awssdk.enhanced.dynamodb.model.*;

import java.util.Map;

public class DynamoDbEnhancedDocumentClientQuery {

    // Create a DynamoDB client with the default settings connected to the DynamoDB
    // endpoint in the default region based on the default credentials provider chain.
    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<EnhancedDocument> DYNAMODB_TABLE =
            ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.documentSchemaBuilder()
                    .addIndexPartitionKey(TableMetadata.primaryIndexName(),"pk",
 AttributeValueType.S)
                    .addIndexSortKey(TableMetadata.primaryIndexName(), "sk",
 AttributeValueType.S)

 .attributeConverterProviders(AttributeConverterProvider.defaultProvider())
                    .build());
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedDocumentClientQuery.class);

    private void query() {
        PageIterable<EnhancedDocument> response =
 DYNAMODB_TABLE.query(QueryEnhancedRequest.builder()
                .filterExpression(Expression.builder()
                        .expression("#name = :name_val")
                        .expressionNames(Map.of("#name", "name"))
                        .expressionValues(Map.of(":name_val",
 AttributeValue.fromS("SomeName")))
                        .build())
                .limit(100)
                .queryConditional(QueryConditional.sortBeginsWith(Key.builder()
                        .partitionValue("id#1")
                        .sortValue("cart#")
                        .build()))
                .scanIndexForward(false)
                .build());

        LOGGER.info("nr of items: " + response.items().stream().count());
        LOGGER.info("First item pk: " +
 response.items().iterator().next().getString("pk"));
```

```
        LOGGER.info("First item sk: " +
 response.items().iterator().next().getString("sk"));

    }
}
```

Example **`Query`** mediante la interfaz de alto nivel

En el siguiente ejemplo, se consulta una tabla denominada `YourTableName` mediante la API de cliente mejorada de DynamoDB.

```
import org.slf4j.*;
import software.amazon.awssdk.enhanced.dynamodb.*;
import software.amazon.awssdk.enhanced.dynamodb.mapper.annotations.*;
import software.amazon.awssdk.enhanced.dynamodb.model.*;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;

import java.util.Map;

public class DynamoDbEnhancedClientQuery {

    private static final DynamoDbEnhancedClient ENHANCED_DYNAMODB_CLIENT =
 DynamoDbEnhancedClient.builder().build();
    private static final DynamoDbTable<YourItem> DYNAMODB_TABLE =
 ENHANCED_DYNAMODB_CLIENT.table("YourTableName",
 TableSchema.fromBean(DynamoDbEnhancedClientQuery.YourItem.class));
    private static final Logger LOGGER =
 LoggerFactory.getLogger(DynamoDbEnhancedClientQuery.class);

    private void query() {
        PageIterable<YourItem> response =
 DYNAMODB_TABLE.query(QueryEnhancedRequest.builder()
                .filterExpression(Expression.builder()
                        .expression("#name = :name_val")
                        .expressionNames(Map.of("#name", "name"))
                        .expressionValues(Map.of(":name_val",
 AttributeValue.fromS("SomeName")))
                        .build())
                .limit(100)
                .queryConditional(QueryConditional.sortBeginsWith(Key.builder()
                        .partitionValue("id#1")
                        .sortValue("cart#")
                        .build()))
```

```
                .scanIndexForward(false)
                .build());

        LOGGER.info("nr of items: " + response.items().stream().count());
        LOGGER.info("First item pk: " + response.items().iterator().next().getPk());
        LOGGER.info("First item sk: " + response.items().iterator().next().getSk());
    }

    @DynamoDbBean
    public static class YourItem {

        public YourItem() {}

        public YourItem(String pk, String sk, String name) {
            this.pk = pk;
            this.sk = sk;
            this.name = name;
        }

        private String pk;
        private String sk;
        private String name;

        @DynamoDbPartitionKey
        public void setPk(String pk) {
            this.pk = pk;
        }

        public String getPk() {
            return pk;
        }

        @DynamoDbSortKey
        public void setSk(String sk) {
            this.sk = sk;
        }

        public String getSk() {
            return sk;
        }

        public void setName(String name) {
            this.name = name;
        }
```

```
        public String getName() {
            return name;
        }
    }
}
```

Interfaz de alto nivel que utiliza clases de datos inmutables

Al realizar una `Query` con las clases de datos inmutables de alto nivel, el código es el mismo que en el ejemplo de la interfaz de alto nivel, excepto en lo que respecta a la construcción de la clase de entidad `YourItem` o `YourImmutableItem`. Consulte el ejemplo PutItem para obtener más información.

Interfaz de alto nivel que utiliza clases de datos inmutables y bibliotecas de generación reutilizables de terceros

Al realizar una `Query` con las clases de datos inmutables de alto nivel, el código es el mismo que en el ejemplo de la interfaz de alto nivel, excepto en lo que respecta a la construcción de la clase de entidad `YourItem` o `YourImmutableLombokItem`. Consulte el ejemplo PutItem para obtener más información.

Ejemplos de código adicionales

Para ver ejemplos adicionales sobre cómo utilizar DynamoDB con el SDK para Java 2.x, consulte los siguientes repositorios de ejemplos de código:

- Ejemplos de código oficiales de una sola acción de AWS
- Ejemplos de código de una sola acción actualizados por la comunidad
- Ejemplos de código oficiales orientados a escenarios de AWS

Programación síncrona y asíncrona

El AWS SDK for Java 2.x proporciona clientes síncronos y asíncronos para Servicios de AWS, como DynamoDB.

Las clases `DynamoDbClient` y `DynamoDbEnhancedClient` proporcionan métodos síncronos que bloquean la ejecución de los subprocesos hasta que el cliente recibe una respuesta del servicio.

Este cliente es la forma más sencilla de interactuar con DynamoDB si no necesita operaciones asíncronas.

Las clases `DynamoDbAsyncClient` y `DynamoDbEnhancedAsyncClient` proporcionan métodos asíncronos que se ejecutan inmediatamente y devuelven el control al subproceso que realiza la llamada sin esperar una respuesta. El cliente sin bloqueo tiene la ventaja de que ofrece una alta simultaneidad en unos pocos subprocesos, lo que permite gestionar de forma eficiente las solicitudes de E/S con pocos recursos de computación. Esto mejora el rendimiento y la capacidad de respuesta.

El AWS SDK for Java 2.x utiliza la compatibilidad nativa con la E/S sin bloqueo. El AWS SDK para Java 1.x tenía que simular la E/S sin bloqueo.

Los métodos síncronos terminan de ejecutarse antes de que haya una respuesta disponible, de manera que necesita una forma de obtener la respuesta cuando esté lista. Los métodos asíncronos en el AWS SDK para Java devuelven un objeto `CompletableFuture` que contiene los resultados de la operación asíncrona en el futuro. Al llamar a `get()` o `join()` en estos objetos `CompletableFuture`, el código se bloquea hasta que el resultado esté disponible. Si los llama al mismo tiempo que realiza la solicitud, el comportamiento es similar al de una llamada síncrona simple.

Para obtener más información sobre la programación asíncrona, consulte Use asynchronous programming en la Guía para desarrolladores de AWS SDK for Java 2.x.

Clientes de HTTP

Para dar soporte a cada cliente, existe un cliente HTTP que se encarga de la comunicación con los Servicios de AWS. Puede conectar clientes HTTP alternativos y elegir uno que tenga las características que mejor se adapten a su aplicación. Algunos son más ligeros y otros tienen más opciones de configuración.

Algunos clientes HTTP solo admiten el uso sincrónico, mientras que otros solo admiten el uso asíncrono. Para ver un diagrama de flujo que le ayude a seleccionar el cliente HTTP óptimo para su carga de trabajo, consulte HTTP client recommendations en la Guía para desarrolladores de AWS SDK for Java 2.x.

La siguiente lista presenta algunos de los posibles clientes HTTP:

Temas

- Cliente HTTP basado en Apache

- Cliente HTTP basado en URLConnection
- Cliente HTTP basado en Netty
- Cliente HTTP basado en CRT de AWS

Cliente HTTP basado en Apache

La clase `ApacheHttpClient` admite clientes de servicios síncronos. Es el cliente HTTP predeterminado para el uso síncrono. Para obtener información sobre la configuración de la clase `ApacheHttpClient`, consulte Configure the Apache-based HTTP client en la Guía para desarrolladores de AWS SDK for Java 2.x.

Cliente HTTP basado en **`URLConnection`**

La clase `UrlConnectionHttpClient` es otra opción para los clientes síncronos. Este cliente se carga más rápido que el cliente HTTP basado en Apache, pero tiene menos características. Para obtener información sobre la configuración de la clase `UrlConnectionHttpClient`, consulte Configure the URLConnection-based HTTP client en la Guía para desarrolladores de AWS SDK for Java 2.x.

Cliente HTTP basado en Netty

La clase `NettyNioAsyncHttpClient` admite clientes asíncronos. Es la opción predeterminada para el uso asíncrono. Para obtener información sobre la configuración de la clase `NettyNioAsyncHttpClient`, consulte Configure the Netty-based HTTP client en la Guía para desarrolladores de AWS SDK for Java 2.x.

Cliente HTTP basado en CRT de AWS

Las nuevas clases `AwsCrtHttpClient` y `AwsCrtAsyncHttpClient` de las bibliotecas AWS Common Runtime (CRT) son otras opciones que admiten clientes síncronos y asíncronos. En comparación con otros clientes HTTP, AWS CRT ofrece:

- Tiempo de inicio del SDK más rápido
- Ocupación de menos espacio de memoria
- Tiempo de latencia reducido
- Administración del estado de conexión

- equilibrio de carga de DNS

Para obtener información sobre la configuración de las clases `AwsCrtHttpClient` y `AwsCrtAsyncHttpClient`, consulte Configure the AWS CRT-based HTTP clients en la Guía para desarrolladores de AWS SDK for Java 2.x.

El cliente HTTP basado en AWS CRT no es el predeterminado porque, de lo contrario, dejaría de ser compatible con versiones anteriores de las aplicaciones. Sin embargo, para DynamoDB, se recomienda utilizar el cliente HTTP basado en AWS CRT para los usos síncrono y asíncrono.

Para leer una introducción sobre el cliente HTTP basado en AWS CRT, consulte Announcing availability of the AWS CRT HTTP Client in the AWS SDK for Java 2.x en el AWS Developer Tools Blog.

Configuración de un cliente HTTP

Al configurar un cliente, puede proporcionar varias opciones de configuración, entre las que se incluyen las siguientes:

- Configurar los tiempos de espera para distintos aspectos de las llamadas a la API.
- Habilitar TCP Keep-Alive.
- Controlar la política de reintentos cuando se producen errores.
- Especificar los atributos de ejecución que pueden modificar las instancias Execution interceptor. Los interceptores de ejecución pueden escribir código que intercepte la ejecución de las solicitudes y respuestas de la API. Esto le permite realizar tareas, como publicar métricas y modificar las solicitudes de forma instantánea.
- Agregar o manipular encabezados HTTP.
- Permitir el seguimiento de las métricas de rendimiento del cliente. Esta característica le permite recopilar métricas sobre los clientes de servicio de su aplicación y analizar los resultados en Amazon CloudWatch.
- Especificar un servicio ejecutor alternativo que se utilizará para programar tareas, como los reintentos asíncronos y las tareas de tiempo de espera.

La configuración se controla proporcionando un objeto `ClientOverrideConfiguration` a la clase `Builder` del cliente de servicio. Verá cómo se hace en algunos ejemplos de código de las siguientes secciones.

La `ClientOverrideConfiguration` proporciona opciones de configuración estándar. Los diferentes clientes HTTP conectables también ofrecen posibilidades de configuración específicas de la implementación.

Temas de esta sección

- Configuración de tiempo de espera
- RetryMode
- DefaultsMode
- Configuración de Keep-Alive

Configuración de tiempo de espera

Puede ajustar la configuración del cliente para controlar varios tiempos de espera relacionados con las llamadas al servicio. DynamoDB proporciona latencias más bajas en comparación con otros Servicios de AWS. Por lo tanto, es posible que desee ajustar estas propiedades para reducir los valores de tiempo de espera, de modo que pueda responder rápido a los errores si se produce un problema de red.

Puede personalizar el comportamiento relacionado con la latencia mediante `ClientOverrideConfiguration` en el cliente DynamoDB o cambiar las opciones de configuración detalladas en la implementación del cliente HTTP subyacente.

Puede configurar las siguientes propiedades impactantes usando `ClientOverrideConfiguration`:

- `apiCallAttemptTimeout`: tiempo que se debe esperar a que se complete un solo intento de una solicitud HTTP antes de abandonar y agotar el tiempo de espera.
- `apiCallTimeout`: tiempo del que dispone el cliente para ejecutar por completo una llamada a la API. Esto incluye la ejecución del controlador de solicitudes, que consta de todas las solicitudes HTTP, incluidos los reintentos.

El AWS SDK for Java 2.x proporciona valores predeterminados para algunas opciones de tiempo de espera, como el tiempo de espera de la conexión y los tiempos de espera del socket. El SDK no proporciona valores predeterminados para los tiempos de espera de las llamadas a la API ni para los tiempos de espera de los intentos de llamadas individuales a la API. Si estos tiempos de espera no están configurados en `ClientOverrideConfiguration`, el SDK utiliza el valor de tiempo

de espera del socket en lugar del tiempo de espera general de las llamadas a la API. El tiempo de espera del socket tiene un valor predeterminado de 30 segundos.

RetryMode

Otra configuración relacionada con la configuración del tiempo de espera que debe tener en cuenta es el objeto de configuración `RetryMode`. Este objeto de configuración contiene un conjunto de comportamientos de reintento.

El SDK para Java 2.x admite los siguientes modos de reintento:

- `legacy`: modo de reintento predeterminado si no lo cambia de forma explícita. Este modo de reintento es específico del SDK de Java. Se caracteriza por hasta tres reintentos, o más para los servicios como DynamoDB, que tiene hasta ocho reintentos.
- `standard`: se denomina “estándar” porque es más consistente con otros AWS SDK. Este modo espera un período de tiempo aleatorio que va desde 0 ms hasta 1000 ms para el primer intento. Si es necesario volver a intentarlo, este modo selecciona otro tiempo aleatorio de 0 ms a 1000 ms y lo multiplica por dos. Si es necesario volver a intentarlo, realiza la misma selección aleatoria multiplicada por cuatro, y así sucesivamente. Cada espera tiene un límite de 20 segundos. Este modo realiza reintentos en caso de que se detecten más condiciones de fallo que el modo `legacy`. En el caso de DynamoDB, realiza hasta un máximo de tres intentos, a menos que se anule con numRetries.
- `adaptive`: se basa en el modo `standard` y limita de forma dinámica la tasa de solicitudes de AWS para maximizar la tasa de éxito. Esto puede producirse a expensas de la latencia de las solicitudes. No recomendamos utilizar el modo de reintento adaptativo cuando la latencia predecible es importante.

En el tema Retry behavior de la Guía de referencia de los SDK y las herramientas de AWS encontrará una definición amplia de estos modos de reintento.

Políticas de reintentos

Todas las configuraciones de `RetryMode` tienen una `RetryPolicy`, que se crea en función de una o más configuraciones de `RetryCondition`. La condición `TokenBucketRetryCondition` es especialmente importante para el comportamiento de reintento de la implementación del cliente del SDK de DynamoDB. Esta condición limita el número de reintentos que realiza el SDK mediante un algoritmo de bucket de tokens. Según el modo de reintento seleccionado, las excepciones de limitación pueden restar o no tokens al `TokenBucket`.

Cuando un cliente detecta un error que se puede volver a intentar, como una excepción de limitación o un error temporal del servidor, el SDK vuelve a intentar la solicitud automáticamente. Puede controlar el número de veces y la rapidez con que se realizan estos reintentos.

Al configurar un cliente, puede proporcionar una `RetryPolicy` que admita los siguientes parámetros:

- `numRetries`: número máximo de reintentos que deben realizarse antes de que una solicitud se considere fallida. El valor predeterminado es ocho, independientemente del modo de reintento que utilice.

  > **Warning**
  >
  > Asegúrese de cambiar este valor predeterminado después de estudiarlo atentamente.

- `backoffStrategy`: la `BackoffStrategy` que se debe aplicar a los reintentos, siendo `FullJitterBackoffStrategy` la estrategia predeterminada. Esta estrategia genera un retardo exponencial entre los reintentos adicionales en función del número o los reintentos actuales, el retardo base y el tiempo máximo de retroceso. A continuación, añade fluctuación para proporcionar un poco de aleatoriedad. El retardo base utilizado en el retardo exponencial es de 25 ms, independientemente del modo de reintento.
- `retryCondition`: la `RetryCondition` determina si se debe volver a intentar una solicitud o no. De forma predeterminada, volverá a intentar un conjunto específico de códigos de estado HTTP y excepciones que considere que se pueden volver a intentar. En la mayoría de las situaciones, la configuración predeterminada debería ser suficiente.

El código siguiente proporciona una política de reintentos alternativa. Además, especifica un total de cinco reintentos (seis solicitudes en total). El primer reintento debe producirse tras un retardo de aproximadamente 100 ms, y cada reintento adicional duplica ese tiempo exponencialmente, hasta un retardo máximo de un segundo.

```
DynamoDbClient client = DynamoDbClient.builder()
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .retryPolicy(RetryPolicy.builder()
            .backoffStrategy(FullJitterBackoffStrategy.builder()
                .baseDelay(Duration.ofMillis(100))
                .maxBackoffTime(Duration.ofSeconds(1))
                .build())
            .numRetries(5)
```

```
            .build())
        .build())
    .build();
```

DefaultsMode

Las propiedades de tiempo de espera que no administran `ClientOverrideConfiguration` y `RetryMode` suelen configurarse implícitamente especificando un `DefaultsMode`.

En AWS SDK for Java 2.x (versión 2.17.102 o posteriores) se introdujo la compatibilidad con `DefaultsMode`. Esta característica proporciona un conjunto de valores predeterminados para los ajustes configurables más comunes, como los ajustes de comunicación HTTP, el comportamiento de los reintentos, los ajustes del punto de conexión regionales del servicio y, potencialmente, cualquier configuración relacionada con el SDK. Al utilizar esta característica, se pueden obtener nuevos valores predeterminados de configuración adaptados a los escenarios de uso habituales.

Los modos predeterminados están estandarizados en todos los SDK de AWS. El SDK para Java 2.x admite los siguientes modos de reintento predeterminados:

- `legacy`: proporciona una configuración predeterminada que varía según el AWS SDK y que existía antes de la creación de `DefaultsMode`.
- `standard`: proporciona una configuración predeterminada no optimizada para la mayoría de los escenarios.
- `in-region`: se basa en el modo estándar e incluye una configuración adaptada a las aplicaciones que llaman a los Servicios de AWS desde la misma Región de AWS.
- `cross-region`: se basa en el modo estándar e incluye una configuración con tiempos de espera elevados para las aplicaciones que llaman a los Servicios de AWS desde una región distinta.
- `mobile`: se basa en el modo estándar e incluye configuraciones con tiempos de espera elevados diseñados para aplicaciones móviles con latencias más altas.
- `auto`: se basa en el modo estándar e incluye funciones experimentales. El SDK intenta descubrir el tiempo de ejecución para determinar automáticamente la configuración adecuada. La detección automática se basa en la heurística y no es precisa al 100 %. Si no se puede determinar el tiempo de ejecución, se utiliza el modo estándar. La detección automática puede consultar los Metadatos de instancia y datos de usuario, lo que puede introducir latencia. Si la startup es fundamental para tu aplicación, te recomendamos que elijas un `DefaultsMode` explícito en su lugar.

Puede configurar el modo predeterminado de las siguientes maneras:

- Directamente en un cliente mediante `AwsClientBuilder.Builder#defaultsMode(DefaultsMode)`
- En un perfil de configuración a través de la propiedad del archivo de perfil `defaults_mode`.
- Globalmente a través de la propiedad del sistema `aws.defaultsMode`.
- Globalmente a través de la variable de entorno `AWS_DEFAULTS_MODE`.

> **Note**
>
> Para cualquier modo que no sea el `legacy`, los valores predeterminados pueden cambiar a medida que evolucionen las prácticas recomendadas. Por lo tanto, recomendamos que se realicen pruebas al actualizar el SDK si se utiliza un modo distinto a `legacy`.

Los valores predeterminados de configuración inteligente de la Guía de referencia de los SDK y las herramientas de AWS proporcionan una lista de las propiedades de configuración y sus valores predeterminados en los distintos modos predeterminados.

El valor de modo predeterminado se elige según las características de la aplicación y el Servicio de AWS con el que interactúa la aplicación.

Estos valores se configuran teniendo en cuenta una amplia selección de Servicios de AWS. Para una implementación típica de DynamoDB en la que tanto las tablas como la aplicación de DynamoDB se implementan en una región, el modo predeterminado `in-region` es el más relevante de los modos predeterminados `standard`.

Example Configuración del cliente del SDK de DynamoDB adaptada para llamadas de baja latencia

En el siguiente ejemplo, los tiempos de espera se adaptan a valores más bajos para una llamada a DynamoDB de baja latencia prevista.

```
DynamoDbAsyncClient asyncClient = DynamoDbAsyncClient.builder()
    .defaultsMode(DefaultsMode.IN_REGION)
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder())
    .overrideConfiguration(ClientOverrideConfiguration.builder()
        .apiCallTimeout(Duration.ofSeconds(3))
        .apiCallAttemptTimeout(Duration.ofMillis(500))
        .build())
    .build();
```

La implementación del cliente HTTP individual puede proporcionar un control aún más detallado sobre el tiempo de espera y el comportamiento de uso de la conexión. Por ejemplo, en el caso del cliente basado en AWS CRT, puede activar `ConnectionHealthConfiguration`, que permite que el cliente monitoree activamente el estado de las conexiones utilizadas. Para obtener más información, consulte Configuración avanzada de clientes HTTP basados en CRT de AWS en la Guía para desarrolladores de AWS SDK for Java 2.x.

Configuración de Keep-Alive

Si se activa keep-alive, se pueden reducir las latencias al reutilizar las conexiones. Hay dos tipos diferentes de keep-alive: HTTP Keep-Alive y TCP Keep-Alive.

- HTTP Keep-Alive intenta mantener la conexión HTTPS entre el cliente y el servidor para que las solicitudes posteriores puedan aprovechar dicha conexión. De este modo, se omite la autenticación HTTPS tan pesada en las solicitudes sucesivas. HTTP Keep-Alive está activado de forma predeterminada en todos los clientes.
- TCP Keep-Alive solicita al sistema operativo subyacente que envíe paquetes pequeños a través de la conexión por socket para garantizar que el socket se mantenga activo y detectar inmediatamente cualquier caída. De este modo, se garantiza que una solicitud posterior no pierda tiempo intentando utilizar un socket caído. TCP Keep-Alive está desactivado de forma predeterminada en todos los clientes. En los siguientes ejemplos de código se muestra cómo activarlo en cada cliente HTTP. Cuando está activado para todos los clientes HTTP no basados en CRT, el mecanismo Keep-Alive real depende del sistema operativo. Por lo tanto, debe configurar valores adicionales de TCP Keep-Alive, como el tiempo de espera y el número de paquetes, a través del sistema operativo. Puede hacerlo con `sysctl` en Linux o macOS, o con los valores del registro en Windows.

Example para activar TCP Keep-Alive en un cliente HTTP basado en Apache

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(ApacheHttpClient.builder().tcpKeepAlive(true))
    .build();
```

Cliente HTTP basado en **`URLConnection`**

Ningún cliente síncrono que use el cliente HTTP basado en `URLConnection` `HttpURLConnection` dispone de ningún mecanismo para activar keep-alive.

Example para activar TCP Keep-Alive en un cliente HTTP basado en Netty

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(NettyNioAsyncHttpClient.builder().tcpKeepAlive(true))
    .build();
```

Example para activar TCP Keep-Alive en un cliente HTTP basado en AWS CRT

Con el cliente HTTP basado en AWS CRT, puede activar TCP Keep-Alive y controlar la duración.

```
DynamoDbClient client = DynamoDbClient.builder()
    .httpClientBuilder(AwsCrtHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

Al utilizar el cliente asíncrono de DynamoDB, puede activar TCP Keep-Alive, tal y como se muestra en el código siguiente.

```
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .httpClientBuilder(AwsCrtAsyncHttpClient.builder()
    .tcpKeepAliveConfiguration(TcpKeepAliveConfiguration.builder()
        .keepAliveInterval(Duration.ofSeconds(50))
        .keepAliveTimeout(Duration.ofSeconds(5))
        .build()))
    .build();
```

Gestión de errores

En lo que respecta a la gestión de excepciones, AWS SDK for Java 2.x utiliza excepciones en tiempo de ejecución (no comprobadas).

La excepción básica, que cubre todas las excepciones del SDK, es `SdkServiceException`, que se extiende desde la `RuntimeException` no comprobada de Java. Si detecta esto, detectará todas las excepciones del SDK.

`SdkServiceException` tiene una subclase llamada `AwsServiceException`. Esta subclase indica cualquier problema de comunicación con el Servicio de AWS. Tiene una subclase llamada `DynamoDbException`, que indica que hay un problema en la comunicación con DynamoDB. Si

detecta esto, detectará todas las excepciones relacionadas con DynamoDB, pero no las demás excepciones del SDK.

En `DynamoDbException` encontrará tipos de excepción más específicos. Algunos de estos tipos de excepción se aplican a las operaciones del plano de control, como `TableAlreadyExistsException`. Otros se aplican a las operaciones del plano de datos. El siguiente es un ejemplo de una excepción común en el plano de datos:

- `ConditionalCheckFailedException`: ha especificado una condición en la solicitud que se ha evaluado como false. Por ejemplo, es posible que haya intentado realizar una actualización condicional de un elemento, pero que el valor real del atributo no coincidiese con el valor previsto en la condición. Una solicitud que falle de esta manera no se volverá a intentar.

Las otras situaciones no tienen una excepción específica definida. Por ejemplo, cuando se limitan las solicitudes, es posible que se lance la `ProvisionedThroughputExceededException` específica, mientras que en otros casos se lanzará la `DynamoDbException` más genérica. En cualquier caso, puede determinar si la excepción se debe a una limitación comprobando si `isThrottlingException()` devuelve `true`.

Según las necesidades de la aplicación, puede detectar todas las instancias `AwsServiceException` o las instancias `DynamoDbException`. Sin embargo, a menudo se necesita un comportamiento diferente en distintas situaciones. La lógica para hacer frente a un fallo en la comprobación de condiciones es diferente a la que se utiliza para gestionar la limitación. Defina las rutas excepcionales con las que quiere trabajar y asegúrese de probar las rutas alternativas. De este modo, se asegurará de que es capaz de enfrentarse a todas las situaciones relevantes.

Consulte Control de errores con DynamoDB para ver una lista de los errores más comunes que puede encontrar. Consulte también Common Errors en la Referencia de la API de Amazon DynamoDB. En la referencia de la API, también se proporcionan los errores exactos posibles para cada operación de la API; por ejemplo, para la operación `Query`. Para obtener información sobre la gestión de excepciones, consulte Exception handling for the AWS SDK for Java 2.x en la Guía para desarrolladores de AWS SDK for Java 2.x.

ID de solicitud de AWS

Cada solicitud incluye un ID de solicitud que puede resultar útil si trabaja con AWS Support para diagnosticar un problema. Cada excepción derivada de `SdkServiceException` tiene un método `requestId()` disponible para recuperar el ID de solicitud.

Registro

El uso del registro proporcionado por el SDK puede resultar útil tanto para detectar cualquier mensaje importante de las bibliotecas de cliente como para realizar una depuración más exhaustiva. Los registradores son jerárquicos. El SDK utiliza `software.amazon.awssdk` como registrador raíz. Puede configurar el nivel con `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `ALL` o `OFF`. El nivel configurado se aplica a ese registrador a la jerarquía de registradores que tiene debajo.

Para el registro, AWS SDK for Java 2.x utiliza Simple Logging Façade for Java (SLF4J). Actúa como una capa de abstracción alrededor de otros registradores y puede usarla para conectar el registrador que prefiera. Para obtener instrucciones sobre cómo conectar los registradores, consulte el SLF4J user manual.

Cada registrador tiene un comportamiento particular. El registrador Log4j 2.x crea un `ConsoleAppender` de forma predeterminada que agrega los eventos de registro a `System.out` y es, de forma predeterminada, el nivel de registro de `ERROR`.

El registrador SimpleLogger incluido en el SLF4J genera `System.err` y va al nivel de registro `INFO` de forma predeterminada.

Recomendamos que establezca el nivel en `WARN` para `software.amazon.awssdk` para todas las implementaciones de producción para detectar cualquier mensaje importante de las bibliotecas de cliente del SDK y, al mismo tiempo, limitar la cantidad de resultados.

Si SLF4J no encuentra un registrador compatible en la ruta de clases (no hay ningún enlace con SLF4J), no implementa ninguna operación de forma predeterminada. Esta implementación hace que se registren mensajes `System.err` que expliquen que SLF4J no ha podido encontrar una implementación de registrador en la ruta de clases. Para evitar esta situación, hay que agregar una implementación de registrador. Para ello, puede añadir una dependencia en el `pom.xml` de Apache Maven en los artefactos, como `org.slf4j.slf4j-simple` o `org.apache.logging.log4j.log4j-slf4j2-imp`.

Para obtener información sobre cómo configurar el registro en el SDK, incluido cómo agregar dependencias de registro a la configuración de la aplicación, consulte Logging with the SDK for Java 2.x en la Guía para desarrolladores de AWS SDK para Java.

La siguiente configuración del archivo `Log4j2.xml` muestra cómo ajustar el comportamiento de registro si se utiliza el registrador Apache Log4j 2. Esta configuración establece el nivel del registrador raíz en `WARN`. Todos los registradores de la jerarquía, incluido el registrador `software.amazon.awssdk`, heredan este nivel de registro.

De forma predeterminada, el resultado es `System.out`. En el siguiente ejemplo, seguimos anulando el appender Log4j de salida predeterminado para aplicar un `PatternLayout` de Log4j personalizado.

Ejemplo de un archivo de configuración de **`Log4j2.xml`**

Esta configuración registra los mensajes en los niveles `ERROR` y `WARN` en la consola para todas las jerarquías de registradores.

```
<Configuration status="WARN">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} [%t] %-5p %c:%L - %m%n" />
    </Console>
  </Appenders>

  <Loggers>
    <Root level="WARN">
      <AppenderRef ref="ConsoleAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

Registro de ID de solicitud de AWS

Si algo sale mal, puede consultar los ID de solicitud dentro de las excepciones. En cambio, si quiere los ID de solicitud para las solicitudes que no generan excepciones, puede usar el registro.

El registrador `software.amazon.awssdk.request` genera los ID de solicitud en el nivel `DEBUG`. El siguiente ejemplo amplía el configuration example anterior para mantener el nivel del registrador raíz en`ERROR`, el `software.amazon.awssdk` en el nivel `WARN` y el `software.amazon.awssdk.request` en el nivel `DEBUG`. Establecer estos niveles ayuda a detectar los ID de solicitud y otros detalles relacionados con la solicitud, como el punto de conexión y el código de estado.

```
<Configuration status="WARN">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{YYYY-MM-dd HH:mm:ss} [%t] %-5p %c:%L - %m%n" />
    </Console>
  </Appenders>
```

```
  <Loggers>
    <Root level="ERROR">
      <AppenderRef ref="ConsoleAppender"/>
    </Root>
    <Logger name="software.amazon.awssdk" level="WARN" />
    <Logger name="software.amazon.awssdk.request" level="DEBUG" />
  </Loggers>
</Configuration>
```

Este es un ejemplo del resultado del registro:

```
2022-09-23 16:02:08 [main] DEBUG software.amazon.awssdk.request:85 - Sending Request:
 DefaultSdkHttpFullRequest(httpMethod=POST, protocol=https, host=dynamodb.us-
east-1.amazonaws.com, encodedPath=/, headers=[amz-sdk-invocation-id, Content-Length,
 Content-Type, User-Agent, X-Amz-Target], queryParameters=[])
 2022-09-23 16:02:08 [main] DEBUG software.amazon.awssdk.request:85 - Received
 successful response: 200, Request ID:
 QS9DUMME2NHEDH8TGT9N5V53OJVV4KQNSO5AEMVJF66Q9ASUAAJG, Extended Request ID: not
 available
```

Paginación

Algunas solicitudes, como `Query` y `Scan`, limitan el tamaño de los datos que se devuelven en una sola solicitud y requieren que realice solicitudes repetidas para abrir las páginas siguientes.

Puede controlar el número máximo de elementos que se debe leer en cada página con el parámetro `Limit`. Por ejemplo, puede usar el parámetro `Limit` para recuperar solo los últimos 10 elementos. Este límite especifica el número de elementos que debe leerse de la tabla antes de aplicar un filtro. Si desea exactamente 10 elementos después de aplicar el filtro, no hay forma de especificarlo. Solo puede controlar el recuento antes de filtrar y comprobarlo desde el cliente cuando haya obtenido los 10 elementos. Las respuestas tienen siempre un tamaño máximo de 1 MB, sin importar el límite.

Es posible que se incluya una `LastEvaluatedKey` en la respuesta de la API. Esto indica que la respuesta ha finalizado porque se ha alcanzado un límite de recuento o un límite de tamaño. Esta clave es la última clave evaluada para esa respuesta. Al interactuar directamente con la API, puede recuperar esta `LastEvaluatedKey` y pasarla a una llamada de seguimiento como `ExclusiveStartKey` para que lea el siguiente fragmento desde ese punto de partida. Si se devuelve ninguna `LastEvaluatedKey`, eso significa que no hay más elementos que coincidan con la llamada a la API `Query` o `Scan`.

En el siguiente ejemplo, se utiliza la interfaz de bajo nivel para limitar los elementos a 100 en función del parámetro `keyConditionExpression`.

```
QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
        .expressionAttributeValues(Map.of(
                ":pk_val", AttributeValue.fromS("123"),
                ":sk_val", AttributeValue.fromN("1000")))
        .keyConditionExpression("pk = :pk_val AND sk > :sk_val")
        .limit(100)
        .tableName(TABLE_NAME);

while (true) {
    QueryResponse queryResponse = DYNAMODB_CLIENT.query(queryRequestBuilder.build());

    queryResponse.items().forEach(item -> {
        LOGGER.info("item PK: [" + item.get("pk") + "] and SK: [" + item.get("sk") +
 "]");
    });

    if (!queryResponse.hasLastEvaluatedKey()) {
        break;
    }
    queryRequestBuilder.exclusiveStartKey(queryResponse.lastEvaluatedKey());
}
```

AWS SDK for Java 2.x puede simplificar esta interacción con DynamoDB al proporcionar métodos de paginación automática que efectúan varias llamadas al servicio para obtener las siguientes páginas de resultados automáticamente. Esto simplifica el código, pero elimina parte del control sobre el uso de los recursos que se mantendría al leer las páginas manualmente.

Al utilizar los métodos `Iterable` disponibles en el cliente de DynamoDB, como `QueryPaginator` y `ScanPaginator`, el SDK se encarga de la paginación. El tipo de retorno de estos métodos es un retorno iterable personalizado que se puede utilizar para iterar en todas las páginas. El SDK gestiona internamente las llamadas al servicio en su nombre. La API de Java Stream permite gestionar el resultado de `QueryPaginator`, tal como se muestra en el siguiente ejemplo.

```
QueryPublisher queryPublisher =
    DYNAMODB_CLIENT.queryPaginator(QueryRequest.builder()
        .expressionAttributeValues(Map.of(
            ":pk_val", AttributeValue.fromS("123"),
            ":sk_val", AttributeValue.fromN("1000")))
```

```
        .keyConditionExpression("pk = :pk_val AND sk > :sk_val")
        .limit(100)
        .tableName("YourTableName")
        .build());

queryPublisher.items().subscribe(item ->
    System.out.println(item.get("itemData"))).join();
```

Anotaciones de clases de datos

El SDK de Java proporciona varias anotaciones que puede incluir en los atributos de la clase de datos. Estas anotaciones influyen en la forma en que el SDK interactúa con los atributos. Al agregar la anotación, puede hacer que un atributo se comporte como un contador atómico implícito, mantener un valor de marca de tiempo generado automáticamente o realizar un seguimiento del número de versión de un elemento. Para obtener más información, consulte Data class annotations.

Control de errores con DynamoDB

En esta sección se describen los errores de tiempo de ejecución y se explica cómo controlarlos. También se describen los mensajes y códigos de error específicos de Amazon DynamoDB. Para obtener una lista de los errores comunes que se aplican a todos los servicios de AWS, consulte Administración de accesos

Temas

- Componentes de un error
- Errores transaccionales
- Mensajes y códigos de error
- Control de errores en la aplicación
- Reintentos de error y retroceso exponencial
- Operaciones por lotes y control de errores

Componentes de un error

Cuando el programa envía una solicitud, DynamoDB intenta procesarla. Si la solicitud se lleva a cabo correctamente, DynamoDB devuelve un código de estado HTTP de operación correcta (`200 OK`), así como el resultado de la operación solicitada.

Si la solicitud no se realiza correctamente, DynamoDB devuelve un error. Cada error tiene tres componentes:

- Un código de estado HTTP (por ejemplo, `400`).
- Un nombre de excepción (por ejemplo, `ResourceNotFoundException`).
- Un mensaje de error (por ejemplo, `Requested resource not found: Table: tablename not found`).

Los SDK de AWS se encargan de transmitir los errores a la aplicación, para que pueda adoptar las medidas apropiadas. Por ejemplo, en un programa en Java, puede escribir una lógica `try-catch` para controlar una excepción `ResourceNotFoundException`.

Si no utiliza un SDK de AWS, tiene que analizar el contenido de la respuesta de bajo nivel de DynamoDB. A continuación se muestra un ejemplo de este tipo de respuesta.

```
HTTP/1.1 400 Bad Request
x-amzn-RequestId: LDM6CJP8RMQ1FHKSC1RBVJFPNVV4KQNSO5AEMF66Q9ASUAAJG
Content-Type: application/x-amz-json-1.0
Content-Length: 240
Date: Thu, 15 Mar 2012 23:56:23 GMT

{"__type":"com.amazonaws.dynamodb.v20120810#ResourceNotFoundException",
"message":"Requested resource not found: Table: tablename not found"}
```

Errores transaccionales

Para obtener información sobre los errores transaccionales, consulte Gestión de conflictos de transacciones en DynamoDB.

Mensajes y códigos de error

A continuación se muestra una lista de excepciones devueltas por DynamoDB, agrupados por su código de estado HTTP. Si ¿Reintentar? es Sí, puede volver a enviar la misma solicitud. Si ¿Reintentar? es No, debe corregir el problema en el lado del cliente antes de volver a enviar la solicitud.

Código de estado HTTP 400

El código de estado HTTP `400` indica que existe un problema en la solicitud; por ejemplo, se ha producido un error de autenticación, faltan parámetros obligatorios o se ha superado el rendimiento aprovisionado de una tabla. Debe corregir el problema en la aplicación antes de volver a enviar la solicitud.

AccessDeniedException

Mensaje: Access denied.

El cliente no firmó correctamente la solicitud. Si utiliza un SDK de AWS, las solicitudes se firman automáticamente; en caso contrario, visite Proceso de firma de Signature Version 4 en la Referencia general de AWS.

¿Reintentar? No

ConditionalCheckFailedException

Mensaje: The conditional request failed.

Ha especificado una condición que se ha evaluado en false. Por ejemplo, es posible que haya intentado realizar una actualización condicional de un elemento, pero que el valor real del atributo no coincidiese con el valor previsto en la condición.

¿Reintentar? No

IncompleteSignatureException

Mensaje: La firma de solicitud no cumple los estándares de AWS).

La firma de la solicitud no incluía todos los componentes necesarios. Si utiliza un SDK de AWS, las solicitudes se firman automáticamente; en caso contrario, visite Proceso de firma de Signature Version 4 en la Referencia general de AWS.

¿Reintentar? No

ItemCollectionSizeLimitExceededException

Mensaje: Collection size exceeded.

Para una tabla con un índice secundario local, un grupo de elementos con el mismo valor de clave de partición ha superado el límite de tamaño máximo de 10 GB. Para obtener más información sobre las colecciones de elementos, consulte Colecciones de elementos en los índices secundarios locales.

¿Reintentar? Sí

LimitExceededException

Mensaje: Too many operations for a given subscriber.

Hay demasiadas operaciones del plano de control simultáneas. El número acumulado de tablas e índices que se encuentren en los estados `CREATING`, `DELETING` o `UPDATING` no puede ser mayor que 500.

¿Reintentar? Sí

MissingAuthenticationTokenException

Mensaje: La solicitud debe contener un ID de clave de acceso de AWS válido (registrado).

La solicitud no incluía el encabezado de autorización necesario o el formato de este último era incorrecto. Consulte API de bajo nivel de DynamoDB.

¿Reintentar? No

ProvisionedThroughputExceededException

Mensaje: You exceeded your maximum allowed provisioned throughput for a table or for one or more global secondary indexes.(Ha excedido el máximo permitido del rendimiento aprovisionado para una tabla con uno o más índices secundario globales) To view performance metrics for provisioned throughput vs. consumed throughput, open the Amazon CloudWatch console (Para ver las métricas de rendimiento para el rendimiento aprovisionado vs. el rendimiento consumido, abra la consola de Amazon CloudWatch).

El error incluye una lista de campos de `ThrottlingReason` que proporciona un contexto específico sobre por qué se produjo la limitación, siguiendo el formato `ResourceType+OperationType+LimitType` (por ejemplo, `TableReadProvisionedThroughputExceeded`) y el ARN del recurso afectado. Esto le ayuda a identificar qué recurso se está limitando (tabla o índice), qué tipo de operación desencadenó

la limitación (lectura o escritura) y el límite específico que se superó (en este caso: capacidad aprovisionada). Para obtener más información sobre el diagnóstico y la resolución de problemas de limitación, consulte Diagnóstico de limitación.

Ejemplo: La velocidad de solicitudes es demasiado alta. Los SDK de AWS para DynamoDB reintentan automáticamente las solicitudes que reciben esta excepción. La solicitud se llevará a cabo con éxito en algún momento, salvo que la cola de reintentos sea demasiado larga para que pueda alcanzarse el final. Reduzca la frecuencia de las solicitudes mediante Reintentos de error y retroceso exponencial.

¿Reintentar? Sí

ReplicatedWriteConflictException

Mensaje: Una solicitud de otra región está modificando uno o más elementos de esta solicitud.

Ejemplo: se solicitó una operación de escritura para un elemento de una tabla global de alta coherencia de varias regiones (MRSC) que actualmente está siendo modificado por una solicitud de otra región.

¿Reintentar? Sí

RequestLimitExceeded

Mensaje: Throughput exceeds the current throughput limit for your account (El rendimiento supera el límite de rendimiento actual de su cuenta). Para solicitar un aumento del límite, contacte con AWS Support en https://aws.amazon.com/support.

El error incluye una lista de campos de `ThrottlingReason` que proporciona un contexto específico sobre por qué se produjo la limitación, siguiendo el formato `ResourceType +OperationType+LimitType` (por ejemplo, `TableWriteAccountLimitExceeded` o `IndexReadAccountLimitExceeded`) y el ARN del recurso afectado. Esto le ayuda a identificar qué recurso se está limitando (tabla o índice), qué tipo de operación desencadenó la limitación (lectura o escritura) y si ha superado las cuotas de servicio por cuenta. Para obtener más información sobre el diagnóstico y la resolución de problemas de limitación, consulte Diagnóstico de limitación.

Ejemplo: La velocidad de las solicitudes bajo demanda excede el rendimiento permitido de la cuenta y la tabla no puede escalarse más.

¿Reintentar? Sí

ResourceInUseException

Mensaje: The resource which you are attempting to change is in use.

Ejemplo: Ha intentado volver a crear una tabla que ya existía o eliminar una tabla que se encuentra en el estado `CREATING`.

¿Reintentar? No

ResourceNotFoundException

Mensaje: Requested resource not found.

Ejemplo: La tabla que se ha solicitado no existe o se encuentra demasiado al principio del estado `CREATING`.

¿Reintentar? No

ThrottlingException

Mensaje: Rate of requests exceeds the allowed throughput.

Esta excepción se devuelve como respuesta de AmazonServiceException con un código de estado THROTTLING_EXCEPTION. Esta excepción es posible que se produzca si realiza operaciones de la API de plano de control con demasiada rapidez.

Para las tablas que utilizan el modo bajo demanda, esta excepción es posible que se produzca para cualquier operación de la API de plano de datos si la tarifa de solicitud es demasiado alta. Para obtener más información sobre el escalado bajo demanda, consulte Rendimiento inicial y propiedades de escalado.

El error incluye una lista de campos de `ThrottlingReason` que proporciona un contexto específico sobre por qué se produjo la limitación, siguiendo el formato `ResourceType+OperationType+LimitType` (por ejemplo, `TableReadKeyRangeThroughputExceeded` o `IndexWriteMaxOnDemandThroughputExceeded`) y el ARN del recurso afectado. Esta información le ayuda a identificar qué recurso se está limitando (tabla o índice), qué tipo de operación desencadenó la limitación (lectura o escritura) y el límite específico que se superó

(límites de partición o rendimiento máximo bajo demanda). Para obtener más información sobre el diagnóstico y la resolución de problemas de limitación, consulte Diagnóstico de limitación.

¿Reintentar? Sí

UnrecognizedClientException

Mensaje: The Access Key ID or security token is invalid.

La firma de la solicitud es incorrecta. La causa más probable es que un ID de clave de acceso o una clave secreta de AWS sean incorrectos.

¿Reintentar? Sí

ValidationException

Mensaje: Varía, según el error concreto de que se trate.

Este error se produce por varias razones; por ejemplo, si se ha omitido un parámetro obligatorio, uno de los valores está fuera del rango admitido o los tipos de datos no concuerdan. El mensaje de error contiene información acerca de la parte concreta de la solicitud que ha provocado el error.

¿Reintentar? No

Código de estado HTTP 5xx

Un código de estado HTTP `5xx` indica un problema cuya resolución corresponde a AWS. Puede ser un error transitorio, en cuyo caso puede reintentar la solicitud hasta que se ejecute satisfactoriamente. En caso contrario, vaya al panel de AWS Service Health Dashboard para comprobar si existe algún problema operativo con el servicio.

Para obtener más información, consulte ¿Cómo resuelvo los errores HTTP 5xx en Amazon DynamoDB?

InternalServerError (HTTP 500)

DynamoDB no pudo procesar la solicitud.

¿Reintentar? Sí

> **Note**
>
> Pueden producirse errores internos del servidor cuando se utilizan elementos. Cabe esperar que esto suceda durante la vida útil de una tabla. Todas las solicitudes que producen un error se pueden reintentar inmediatamente.
> Cuando se recibe un código de estado 500 en una operación de escritura, es posible que la operación se haya realizado correctamente o haya fallado. Si la operación de escritura es una solicitud `TransactWriteItem`, entonces puede volver a intentar la operación. Si la operación de escritura es una solicitud de escritura de un solo elemento, como `PutItem`, `UpdateItem` o `DeleteItem`, la aplicación debe leer el estado del elemento antes de volver a intentar la operación o utilizar Ejemplo de la CLI de expresión de condición de DynamoDB para asegurarse de que el elemento permanezca en el estado correcto después de volver a intentarlo, independientemente de si la operación anterior se ha realizado correctamente o ha fallado. Si la idempotencia es un requisito para la operación de escritura, utilice `TransactWriteItem`, que admite solicitudes idempotentes especificando automáticamente un `ClientRequestToken` para desambiguar múltiples intentos de realizar la misma acción.

ServiceUnavailable (HTTP 503)

DynamoDB no está disponible en este momento. Debería tratarse de un estado temporal.

¿Reintentar? Sí

Control de errores en la aplicación

Para que la aplicación funcione sin problemas, es preciso agregar lógica que capture los errores y responda a ellos. Los enfoques habituales incluyen el uso de bloques `try-catch` o instrucciones `if-then`.

Los SDK de AWS llevan a cabo sus propios reintentos y comprobaciones de errores. Si se produce algún error al utilizar uno de los SDK de AWS, su código y descripción pueden ayudarle a solucionar el problema.

También debería aparecer el `Request ID` en la respuesta. El `Request ID` puede resultar útil si tiene que acudir a Support de AWS para diagnosticar el problema.

Reintentos de error y retroceso exponencial

Numerosos componentes de una red, como los servidores DNS, los conmutadores o los balanceadores de carga, entre otros, pueden generar errores en cualquier punto de la vida de una solicitud determinada. La técnica habitual para abordar estas respuestas de error en un entorno de red consiste en implementar los reintentos en la aplicación cliente. Esta técnica aumenta la fiabilidad de la aplicación.

Cada SDK de AWS implementa automáticamente una lógica de reintento. Puede modificar los parámetros de reintento de acuerdo con sus necesidades. Por ejemplo, tomemos una aplicación en Java que requiere una estrategia de conmutación por error rápida y no permite reintentos en caso de error. Con el AWS SDK para Java, podría usar la clase `ClientConfiguration` y proporcionar un valor de `maxErrorRetry` de `0` para desactivar los reintentos. Para obtener más información, consulte la documentación del SDK de AWS del lenguaje de programación específico.

Si no utiliza un SDK de AWS, debe reintentar las solicitudes originales que reciban errores de servidor (5xx). Sin embargo, los errores de cliente (4xx, salvo las excepciones `ThrottlingException` o `ProvisionedThroughputExceededException`) indican que es preciso revisar la solicitud en sí para corregir el problema antes de volver a intentarlo. Para obtener recomendaciones detalladas para abordar situaciones de limitación específicas, consulte la sección de solución de problemas de limitación de DynamoDB.

Además de los reintentos sencillos, cada SDK de AWS implementa un algoritmo de retroceso exponencial para mejorar el control de flujo. El retardo exponencial se basa en el concepto de utilizar tiempos de espera progresivamente más largos entre reintentos para las respuestas a errores consecutivos. Por ejemplo, las aplicaciones cliente podrían esperar 50 milisegundos antes de llevar a cabo el primer reintento, 100 milisegundos antes del segundo, hasta 200 milisegundos antes del tercero y así sucesivamente. Sin embargo, si la solicitud no se ha llevado a cabo correctamente al cabo de un minuto, el problema podría radicar en que el tamaño de la solicitud supera el rendimiento aprovisionado, y no en la tasa de solicitudes. Establezca el número máximo de reintentos de modo que se detenga al cabo de un minuto. Si la solicitud no se realiza correctamente, investigue las opciones de rendimiento aprovisionado.

> **Note**
>
> Los SDK de AWS implementan una lógica de reintento automático y retroceso exponencial.

La mayoría los algoritmos de retardo exponencial utilizan la fluctuación (el retraso aleatorio) para evitar conflictos sucesivos. Habida cuenta de que no está intentando evitar este tipo de conflictos en estos casos, no es preciso utilizar este número aleatorio. Sin embargo, si utiliza clientes simultáneos, la fluctuación puede ayudar a que las solicitudes tengan éxito con mayor rapidez. Para obtener más información, consulte la entrada del blog sobre Exponential Backoff and Jitter (Retroceso exponencial y fluctuación).

Operaciones por lotes y control de errores

La API de bajo nivel de DynamoDB admite las operaciones por lote de lectura y escritura. `BatchGetItem` lee elementos en una o varias tablas y `BatchWriteItem` coloca o elimina elementos en una o varias tablas. Estas operaciones por lote se implementan como encapsuladores en torno a otras operaciones de DynamoDB que no son por lote. Es decir, `BatchGetItem` invoca `GetItem` una vez para cada elemento del lote. De igual forma,`BatchWriteItem` invoca `DeleteItem` o `PutItem`, según proceda, para cada elemento del lote.

Una operación por lotes puede tolerar que algunas solicitudes individuales del lote no se lleven a cabo. Por ejemplo, tomemos una solicitud `BatchGetItem` para leer cinco elementos. Aunque algunas de las solicitudes `GetItem` subyacentes no se realicen, esto no provocará un error de toda la operación `BatchGetItem`. Sin embargo, si las cinco operaciones de lectura fallan, entonces el `BatchGetItem` completo falla.

Las operaciones por lotes devuelven información sobre las solicitudes individuales que no se realizan, para que pueda diagnosticar el problema y reintentar la operación. Para `BatchGetItem`, las tablas y claves principales en cuestión se devuelven en el valor `UnprocessedKeys` de la respuesta. Para `BatchWriteItem`, se devuelve información similar en `UnprocessedItems`.

La causa más probable de que no se realice una lectura o una escritura es la limitación controlada. Para `BatchGetItem`, una o varias tablas de la solicitud por lotes no tiene suficiente capacidad de lectura aprovisionada para admitir la operación. Para `BatchWriteItem`, una o varias de las tablas no tiene suficiente capacidad de escritura aprovisionada.

Si DynamoDB devuelve elementos sin procesar, debe reintentar la operación por lote para estos elementos. Sin embargo, recomendamos encarecidamente utilizar un algoritmo de retardo exponencial. Si reintenta la operación de forma inmediata, podría volver a producirse un error en las solicitudes subyacentes de lectura o escritura a causa de la limitación controlada de las tablas individuales. Si retrasa la operación por lotes mediante el retardo exponencial, será mucho más probable que las solicitudes individuales del lote se lleven a cabo correctamente.

Uso de DynamoDB con un SDK de AWS

Los kits de desarrollo de software (SDK) de AWS se encuentran disponibles en muchos lenguajes de programación populares. Cada SDK proporciona una API, ejemplos de código y documentación que facilitan a los desarrolladores la creación de aplicaciones en su lenguaje preferido.

Documentación de SDK	Ejemplos de código
AWS SDK para C++	Ejemplos de código de la AWS SDK para C++
AWS CLI	Ejemplos de código de la AWS CLI
AWS SDK para Go	Ejemplos de código de la AWS SDK para Go
AWS SDK para Java	Ejemplos de código de la AWS SDK para Java
AWS SDK para JavaScript	Ejemplos de código de la AWS SDK para JavaScript
AWS SDK para Kotlin	Ejemplos de código de la AWS SDK para Kotlin
AWS SDK para .NET	Ejemplos de código de la AWS SDK para .NET
AWS SDK para PHP	Ejemplos de código de la AWS SDK para PHP
Herramientas de AWS para PowerShell	Ejemplos de código de la Herramientas de AWS para PowerShell
AWS SDK for Python (Boto3)	Ejemplos de código de la AWS SDK for Python (Boto3)
AWS SDK para Ruby	Ejemplos de código de la AWS SDK para Ruby
AWS SDK para Rust	Ejemplos de código de la AWS SDK para Rust
AWS SDK for SAP ABAP	Ejemplos de código de la AWS SDK for SAP ABAP
AWS SDK para Swift	Ejemplos de código de la AWS SDK para Swift

Para ver ejemplos específicos de DynamoDB, consulte Ejemplos de código de DynamoDB con los SDK de AWS.

> **Ejemplo de disponibilidad**
>
> ¿No encuentra lo que necesita? Solicite un ejemplo de código a través del enlace de Enviar comentarios que se encuentra al final de esta página.

Uso de tablas, elementos, consultas, análisis e índices

Esta sección proporciona detalles sobre el uso de tablas, elementos, consultas y mucho más en Amazon DynamoDB.

Temas

- Uso de tablas y datos en DynamoDB
- Tablas globales: multiactivas, replicación en varias regiones
- Uso de elementos y atributos en DynamoDB
- Mejora del acceso con índices secundarios en DynamoDB
- Administración de flujos de trabajo complejos con transacciones de DynamoDB
- Captura de datos de cambio con Amazon DynamoDB

Uso de tablas y datos en DynamoDB

En esta sección se describe cómo usar la AWS Command Line Interface (AWS CLI) y los SDK de AWS para crear, actualizar y eliminar tablas de Amazon DynamoDB.

> **Note**
>
> También puede llevar a cabo las mismas tareas utilizando la Consola de administración de AWS. Para obtener más información, consulte Uso de la consola.

En esta sección también se ofrece más información sobre la capacidad de rendimiento, el uso del escalado automático de DynamoDB o la configuración manual del rendimiento aprovisionado.

Temas

- Operaciones básicas en tablas de DynamoDB
- Aspectos que tener en cuenta a la hora de elegir una clase de tabla en DynamoDB
- Cómo agregar etiquetas a los recursos en DynamoDB

Operaciones básicas en tablas de DynamoDB

Al igual que otros sistemas de base de datos, Amazon DynamoDB almacena datos en tablas. Puede administrar las tablas con una pocas operaciones básicas.

Temas

- Creación de una tabla
- Descripción de una tabla
- Actualización de una tabla
- Eliminación de una tabla
- Uso de la protección contra eliminación
- Enumeración de nombres de tablas
- Descripción de las cuotas de rendimiento aprovisionado

Creación de una tabla

Use la operación `CreateTable` para crear una tabla en Amazon DynamoDB. Para crear la tabla, debe proporcionar la siguiente información:

- Nombre de la tabla. El nombre debe cumplir las reglas de nomenclatura de DynamoDB y debe ser único para la región y cuenta de AWS actuales. Por ejemplo, puede crear una tabla `People` en EE. UU. Este (Norte de Virginia) y otra tabla `People` en Europa (Irlanda). Sin embargo, estas dos tablas serían totalmente distintas entre sí. Para obtener más información, consulte Tipos de datos y reglas de nomenclatura admitidos en Amazon DynamoDB.
- Clave principal. La clave principal puede constar de un atributo (clave de partición) o dos (clave de partición y clave de ordenación). Debe proporcionar el nombre, el tipo de datos y el rol de cada atributo: `HASH` (para una clave de partición) y `RANGE` (para una clave de ordenación). Para obtener más información, consulte Clave principal.
- Ajustes de rendimiento (de las tablas aprovisionadas). Si utiliza el modo aprovisionado, debe especificar los ajustes iniciales de rendimiento de lectura y escritura de la tabla. Puede modificar esta configuración en otro momento o habilitar la función Auto Scaling de DynamoDB para administrarlos automáticamente. Para obtener más información, consulte Modo de capacidad aprovisionada de DynamoDB y Administración automática de la capacidad de rendimiento con la función Auto Scaling de DynamoDB.

Ejemplo 1: creación de una tabla bajo demanda

Para crear la misma tabla `Music` utilizando el modo bajo demanda:

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode=PAY_PER_REQUEST
```

La operación `CreateTable` devuelve metadatos de la tabla, como se muestra a continuación:

```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 0,
            "ReadCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        },
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
```

```
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397468.348
    }
}
```

> **Important**
>
> Al llamar a `DescribeTable` en una tabla bajo demanda, las unidades de capacidad de lectura y de escritura se establecen en 0.

Ejemplo 2: creación de una tabla aprovisionada

En el ejemplo siguiente de la AWS CLI se muestra cómo crear una tabla (`Music`). La clave principal consta de `Artist` (clave de partición) y `SongTitle` (clave de ordenación), ambas de tipo `String`. El desempeño máximo de esta tabla es de 10 unidades de capacidad de lectura y 5 unidades de capacidad de escritura.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5
```

La operación `CreateTable` devuelve metadatos de la tabla, como se muestra a continuación:

```
{
    "TableDescription": {
```

```
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 10
        },
        "TableSizeBytes": 0,
        "TableName": "Music",
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397215.37
    }
}
```

El componente `TableStatus` indica el estado actual de la tabla (`CREATING`). Puede que la tabla tarde un tiempo en crearse, según los valores especificados para `ReadCapacityUnits` y `WriteCapacityUnits`. Cuanto mayores sean estos valores, más recursos tendrá que asignar DynamoDB a la tabla.

Ejemplo 3: creación de una tabla mediante la clase de tabla de acceso poco frecuente estándar de DynamoDB

Para crear lo misma tabla `Music` utilizando la clase de tabla de acceso DynamoDB Estándar - Acceso poco frecuente.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

La operación `CreateTable` devuelve metadatos de la tabla, como se muestra a continuación:

```
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music",
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 10
        },
        "TableClassSummary": {
            "LastUpdateDateTime": 1542397215.37,
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "TableSizeBytes": 0,
```

```
        "TableName": "Music",
        "TableStatus": "CREATING",
        "TableId": "12345678-0123-4567-a123-abcdefghijkl",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1542397215.37
    }
}
```

Descripción de una tabla

Para ver información detallada sobre una tabla, utilice la operación `DescribeTable`. Debe proporcionar el nombre de la tabla. El resultado de `DescribeTable` tiene el mismo formato que el de `CreateTable`; Incluye la marca temporal del momento de creación de la tabla, su esquema de claves, su configuración de rendimiento aprovisionado, su tamaño aproximado y los índices secundarios (si los hay).

> **Important**
>
> Al llamar a `DescribeTable` en una tabla bajo demanda, las unidades de capacidad de lectura y de escritura se establecen en 0.

Example

```
aws dynamodb describe-table --table-name Music
```

La tabla está lista para usarla cuando el valor de `TableStatus` cambia de `CREATING` a `ACTIVE`.

> **Note**
>
> Si emite una solicitud `DescribeTable` inmediatamente después de una solicitud `CreateTable`, DynamoDB podría devolver un error (`ResourceNotFoundException`). El motivo es que `DescribeTable` usa una consulta con consistencia final, aunque los metadatos de la tabla podrían no estar disponibles todavía. Espere unos segundos y repita la solicitud `DescribeTable`.
>
> A efectos de facturación, los costes de almacenamiento de DynamoDB incluyen un importe por elemento de 100 bytes en concepto de gastos generales. (Para obtener más información, consulte los precios de DynamoDB). Este importe adicional de 100 bytes por elemento no se utiliza al calcular las unidades de capacidad ni en la operación `DescribeTable`.

Actualización de una tabla

La operación `UpdateTable` permite realizar una de las acciones siguientes:

- Modificar los ajustes de rendimiento aprovisionado de una tabla (para las tablas en modo aprovisionado).
- Cambiar el modo de capacidad de lectura/escritura de la tabla.
- Manipular los índices secundarios globales de la tabla (consulte Uso de índices secundarios globales en DynamoDB).
- Habilite o desactive los DynamoDB Streams en la tabla (consulte Captura de datos de cambios para DynamoDB Streams).

Example

En el siguiente ejemplo de la AWS CLI se muestra cómo modificar los ajustes de desempeño provisionado de una tabla:

```
aws dynamodb update-table --table-name Music \
    --provisioned-throughput ReadCapacityUnits=20,WriteCapacityUnits=10
```

> **Note**
>
> Cuando se emite una solicitud `UpdateTable`, el estado de la tabla cambia de `AVAILABLE` a `UPDATING`. La tabla permanece plenamente disponible para su uso mientras se encuentra

en el estado `UPDATING`. Cuando este proceso finaliza, el estado de la tabla cambia de `UPDATING` a `AVAILABLE`.

Example

En el siguiente ejemplo de la AWS CLI se muestra cómo modificar el modo de capacidad de lectura/escritura de una tabla al modo bajo demanda:

```
aws dynamodb update-table --table-name Music \
    --billing-mode PAY_PER_REQUEST
```

Eliminación de una tabla

Puede eliminar las tablas que no utilice con la operación `DeleteTable`. La eliminación de una tabla es una operación irrecuperable. Para eliminar una tabla con la Consola de administración de AWS, consulte the section called “Paso 6: (Opcional) Limpieza”.

Example

En el ejemplo siguiente de AWS CLI, se muestra cómo se elimina una tabla.

```
aws dynamodb delete-table --table-name Music
```

Cuando se emite una solicitud `DeleteTable`, el estado de la tabla cambia de `ACTIVE` a `DELETING`. Puede que la tabla tarde un tiempo en eliminarse, según los recursos que utilice; por ejemplo, los datos almacenados en la tabla y las secuencias o índices que contenga.

Cuando la operación `DeleteTable` concluya, la tabla ya no existirá en DynamoDB.

Uso de la protección contra eliminación

Puede proteger una tabla contra la eliminación accidental con la propiedad de protección contra la eliminación. Activar esta propiedad para las tablas contribuye a garantizar que las tablas no se eliminen accidentalmente durante las operaciones habituales de administración de tablas por parte de sus administradores. De este modo evitará que se interrumpan las operaciones de negocio normales.

El propietario de la tabla o un administrador autorizado controla la propiedad de protección contra la eliminación de cada tabla. La propiedad de protección contra eliminación para cada tabla está

desactivada de forma predeterminada. Se incluyen las réplicas globales y las tablas restauradas a partir de copias de seguridad. Cuando la protección contra el borrado está desactivada para una tabla, ésta puede ser eliminada por cualquier usuario autorizado por una política de Identity and Access Management (IAM). Cuando la protección contra la eliminación está activada para una tabla, nadie puede eliminarla.

Para cambiar esta configuración, vaya a la configuración adicional de la tabla, navegue al panel Protección contra eliminación y seleccione Habilitar protección contra eliminaciones.

La propiedad de protección contra eliminación se admite en la consola de DynamoDB, API, CLI/SDK y CloudFormation. La API `CreateTable` admite la propiedad de protección contra eliminación en el momento de creación de la tabla y la API `UpdateTable` admite el cambio de la propiedad de protección contra eliminación para las tablas existentes.

> **Note**
>
> - Si se elimina una cuenta de AWS, todos los datos de esa cuenta, incluidas las tablas, se eliminan igualmente en un plazo de 90 días.
> - Si DynamoDB pierde el acceso a una clave administrada por el cliente que se utilizó para cifrar una tabla, seguirá archivando la tabla. Archivar implica hacer una copia de seguridad de la tabla y eliminar el original.

Enumeración de nombres de tablas

La operación `ListTables` devuelve los nombres de las tablas de DynamoDB de la región y cuenta de AWS actuales.

Example

En el siguiente ejemplo de la AWS CLI se muestra cómo enumerar los nombres de las tablas de DynamoDB.

```
aws dynamodb list-tables
```

Descripción de las cuotas de rendimiento aprovisionado

La operación `DescribeLimits` devuelve las cuotas vigentes de capacidad de lectura y escritura de la región y cuenta de AWS actuales.

Example

En el siguiente ejemplo de la AWS CLI se muestra cómo describir las cuotas de rendimiento aprovisionado actuales.

```
aws dynamodb describe-limits
```

El resultado muestra las cuotas superiores de unidades de capacidad de lectura y escritura de la región y cuenta de AWS actuales.

Para obtener más información acerca de estas cuotas y cómo solicitar un aumento de cuotas, consulte Cuotas de rendimiento predeterminadas.

Aspectos que tener en cuenta a la hora de elegir una clase de tabla en DynamoDB

DynamoDB ofrece dos clases de tablas diseñadas para ayudarle a optimizar los costos. La clase de tabla DynamoDB Estándar es la predeterminada y se recomienda para la gran mayoría de las cargas de trabajo. La clase de tabla DynamoDB Estándar - Acceso poco frecuente (DynamoDB Standard-IA) está optimizada para tablas en las que el almacenamiento es el costo dominante. Por ejemplo, las tablas que almacenan datos a los que se accede con poca frecuencia, como registros de aplicaciones, publicaciones antiguas en redes sociales, historial de pedidos de comercio electrónico y antiguos logros en juegos, son buenas candidatas para la clase de tabla Standard - IA.

Cada tabla de DynamoDB está asociada a una clase de tabla. Todos los índices secundarios asociados a la tabla utilizan la misma clase de tabla. Puede configurar la clase de tabla al crear la tabla (DynamoDB Estándar de forma predeterminada) y actualizar la clase de tabla de una tabla existente mediante la Consola de administración de AWS, la CLI de AWS o el SDK de AWS. DynamoDB también es compatible con la administración de la clase de tabla mediante AWS CloudFormation para tablas de una región (tablas que no son globales). Cada clase de tabla ofrece diferentes precios para el almacenamiento de datos, así como para las solicitudes de lectura y escritura. A la hora de elegir una clase de tabla para su tabla, tenga en cuenta lo siguiente:

- La clase de tabla DynamoDB Estándar ofrece menores costos de rendimiento que DynamoDB Standard - IA y es la opción más rentable para tablas en las que el rendimiento es el costo dominante.
- La clase de tabla DynamoDB Standard - IA ofrece menores costos de almacenamiento que DynamoDB Estándar y es la opción más rentable para tablas en las que el almacenamiento es el

costo dominante. Cuando el almacenamiento supera el 50 % del coste de rendimiento (lecturas y escrituras) de una tabla que utiliza la clase de tabla DynamoDB Estándar, la clase de tabla DynamoDB Standard - IA puede ayudarlo a reducir el costo total de la tabla.

- Las tablas DynamoDB Standard-IA ofrecen el mismo rendimiento, durabilidad y disponibilidad que las tablas DynamoDB Standard.
- El cambio entre las clases de tabla DynamoDB Standard y DynamoDB Standard-IA no requiere cambiar el código de la aplicación. Se utilizan las mismas API de DynamoDB y punto de conexión de servicio, independientemente de la clase de tabla que utilicen sus tablas.
- Las tablas DynamoDB Standard-IA son compatibles con todas las características existentes de DynamoDB, tales como escalado automático, modo bajo demanda, período de vida (TTL), copias de seguridad bajo demanda, recuperación a un momento dado (PITR) e índices secundarios globales.

La clase de tabla más rentable para su tabla depende de los patrones de uso de almacenamiento y rendimiento previstos para su tabla. Puede consultar el historial de costos de almacenamiento y rendimiento de la tabla y su uso con los Informes de costos y uso de AWS y con AWS Cost Explorer. Utilice estos datos históricos para determinar la clase de tabla más rentable para su tabla. Para obtener más información sobre el uso de los Informes de costos y uso de AWS y AWS Cost Explorer, consulte la Documentación sobre Billing and Cost Management de AWS. Consulte Precio de Amazon DynamoDB para obtener más información sobre los precios de las clases de tablas.

> **Note**
>
> Una actualización de clase de tabla es un proceso en segundo plano. Puede seguir accediendo a la tabla con normalidad durante la actualización de una clase de tabla. El tiempo de actualización de la clase de tabla depende del tráfico de la tabla, el tamaño del almacenamiento y otras variables relacionadas. No se permiten más de dos actualizaciones de la clase de tabla en un período de 30 días.

Cómo agregar etiquetas a los recursos en DynamoDB

Puede asignar etiquetas a los recursos de Amazon DynamoDB. Las etiquetas le permiten categorizar los recursos de distintas maneras; por ejemplo, según su finalidad, propietario, entorno u otro criterio. Las etiquetas pueden ayudar a hacer lo siguiente:

- Identificar rápidamente un recurso según las etiquetas que le haya asignado.

- Consultar las facturas de AWS desglosadas por etiquetas.

> **Note**
>
> A los índices secundarios locales (LSI) y los índices secundarios globales (GSI) relacionados con las tablas etiquetadas se les asignan automáticamente las mismas etiquetas. En la actualidad, no se puede etiquetar el uso de DynamoDB Streams.

El etiquetado es compatible con servicios de AWS como Amazon EC2, Amazon S3, DynamoDB, entre otros. Un etiquetado eficiente puede ofrecerle información detallada sobre los costos, ya que le permite crear informes sobre los servicios que llevan una etiqueta determinada.

Para empezar a usar etiquetas, haga lo siguiente:

1. Comprender Restricciones de etiquetado en DynamoDB.
2. Crear etiquetas mediante Recursos de etiquetado en DynamoDB.
3. Usar Uso de etiquetas de DynamoDB para crear informes de asignación de costos para realizar el seguimiento de los costes de AWS según su etiqueta activa.

Por último, es conveniente seguir estrategias de etiquetado óptimas. Para obtener más información, consulte Estrategias de etiquetado de AWS.

Restricciones de etiquetado en DynamoDB

Cada etiqueta consta de una clave y un valor, ambos definidos por el usuario. Se aplican las siguientes restricciones:

- Cada tabla puede de DynamoDB solo puede tener una etiqueta con la misma clave. Si intenta agregar una etiqueta existente (con la misma clave), el valor de la etiqueta existente se actualiza con el valor nuevo.
- Las claves y los valores de las etiquetas distinguen entre mayúsculas y minúsculas.
- La longitud máxima de la clave es de 128 caracteres Unicode.
- La longitud máxima del valor es de 256 caracteres Unicode.
- Los caracteres permitidos son letras, espacios en blanco y números, además de los caracteres especiales siguientes: `+ - = . _ : /`
- El número máximo de etiquetas por recurso es 50.

- El tamaño máximo admitido para todas las etiquetas de una tabla es de 10 KB.
- A los nombres y valores de etiquetas asignados por AWS se les asigna automáticamente el prefijo `aws:`, que no puede asignar. Los nombres de etiquetas asignados por AWS no cuentan para el límite de etiquetas de 50 o el límite de tamaño máximo de 10 KB. Los nombres de etiquetas asignados por el usuario presentan el prefijo `user:` en el informe de asignación de costos.
- No es posible antedatar la aplicación de una etiqueta.

Recursos de etiquetado en DynamoDB

Puede usar la consola de Amazon DynamoDB o la AWS Command Line Interface (AWS CLI) para añadir, enumerar, editar o eliminar etiquetas. A continuación, puede activar estas etiquetas definidas por el usuario de modo que aparezcan en la consola de Administración de facturación y costos de AWS y así poder usarlas para el seguimiento de asignación de costos. Para obtener más información, consulte Uso de etiquetas de DynamoDB para crear informes de asignación de costos.

Para la edición en bloque, también puede usar Tag Editor en la Consola de administración de AWS. Para obtener más información, consulte Uso de Tag Editor.

Para usar la API de DynamoDB en su lugar, consulte las siguientes operaciones en la Referencia de la API de Amazon DynamoDB:

- TagResource
- UntagResource
- ListTagsOfResource

Temas

- Establecer permisos para filtrar por etiquetas
- Añadir etiquetas a tablas nuevas o existentes (Consola de administración de AWS)
- Añadir etiquetas a tablas nuevas o existentes (AWS CLI)

Establecer permisos para filtrar por etiquetas

Para utilizar etiquetas a fin de filtrar la lista de tablas en la consola de DynamoDB, asegúrese de que las políticas del usuario incluyan acceso a las siguientes operaciones:

- `tag:GetTagKeys`

- `tag:GetTagValues`

Puede acceder a estas operaciones adjuntando una nueva política de IAM a su usuario siguiendo los pasos que se indican a continuación.

1. Ingrese a la Consola de IAM con un usuario administrador.
2. Seleccione “Policies” (Políticas) en el menú de navegación izquierdo.
3. Seleccione “Create Policy” (Crear política).
4. Pegue la siguiente política en el editor JSON.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "tag:GetTagKeys",
                "tag:GetTagValues"
            ],
            "Resource": "*"
        }
    ]
}
```

5. Complete el asistente y asigne un nombre a la política (por ejemplo, `TagKeysAndValuesReadAccess`).
6. Seleccione “Users” (Usuarios) en el menú de navegación izquierdo.
7. En la lista, seleccione el usuario que utiliza habitualmente para acceder a la consola de DynamoDB.
8. Seleccione “Add permissions” (Añadir permisos).
9. Seleccione “Attach existing policies directly” (Asociar directamente las políticas existentes).
10. En la lista, seleccione la política que ha creado previamente.
11. Complete el asistente.

Añadir etiquetas a tablas nuevas o existentes (Consola de administración de AWS)

Puede usar la consola de DynamoDB para añadir etiquetas a tablas nuevas cuando las cree, o para añadir, editar o eliminar etiquetas de tablas existentes.

Para etiquetar recursos al crearlos (consola)

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación, elija Tablas y, a continuación, seleccione Crear tabla.
3. En la página Create DynamoDB table (Crear tabla de DynamoDB), proporcione un nombre y una clave principal. En la sección Tags (Etiquetas), elija Add new tag (Agregar nueva etiqueta) e ingrese las etiquetas que desee utilizar.

   Para obtener más información sobre la estructura de las etiquetas, consulte Restricciones de etiquetado en DynamoDB.

   Para obtener más información sobre cómo crear tablas, consulte Operaciones básicas en tablas de DynamoDB.

Para etiquetar recursos existentes (consola)

Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.

1. En el panel de navegación, elija Tablas.
2. Elija una tabla de la lista y, a continuación, elija la pestaña Additional settings (Configuración adicional). Puede agregar, editar o eliminar las etiquetas en la sección Tags (Etiquetas) en la parte inferior de la página.

Añadir etiquetas a tablas nuevas o existentes (AWS CLI)

En los ejemplos siguientes se muestra cómo usar la AWS CLI para especificar etiquetas al crear tablas e índices, así como para etiquetar recursos existentes.

Para etiquetar recursos al crearlos (AWS CLI)

- En el ejemplo siguiente se crea una nueva tabla de `Movies` y se añade la etiqueta `Owner` con un valor de `blueTeam`:

```
aws dynamodb create-table \
    --table-name Movies \
    --attribute-definitions AttributeName=Title,AttributeType=S \
    --key-schema AttributeName=Title,KeyType=HASH \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

Para etiquetar recursos existentes (AWS CLI)

- En el ejemplo siguiente se añade la etiqueta `Owner` con un valor de `blueTeam` para la tabla `Movies`:

```
aws dynamodb tag-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies \
    --tags Key=Owner,Value=blueTeam
```

Para enumerar todas las etiquetas de una tabla (AWS CLI)

- En el ejemplo siguiente se muestran todas las etiquetas asociadas con la tabla `Movies`:

```
aws dynamodb list-tags-of-resource \
    --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/Movies
```

Uso de etiquetas de DynamoDB para crear informes de asignación de costos

AWS utiliza etiquetas para organizar los costos de los recursos en el informe de asignación de costos. AWS proporciona dos tipos de etiquetas de asignación de costos:

- Una etiqueta generada por AWS. AWS define, crea y aplica esta etiqueta por usted.
- Etiquetas definidas por el usuario. Puede definir, crear y aplicar estas etiquetas.

Debe activar ambos tipos de etiquetas por separado para que puedan aparecer en el Explorador de costos o en un informe de asignación de costos.

Para activar las etiquetas generadas por AWS:

1. Inicie sesión en la Consola de administración de AWS abra la consola de Billing and Cost Management en https://console.aws.amazon.com/billing/home#/.
2. En el panel de navegación, seleccione Etiquetas de asignación de costos.
3. En Etiquetas de asignación de costos generada por AWS, elija Activar.

Para activar las etiquetas definidas por el usuario:

1. Inicie sesión en la Consola de administración de AWS abra la consola de Billing and Cost Management en https://console.aws.amazon.com/billing/home#/.
2. En el panel de navegación, seleccione Etiquetas de asignación de costos.
3. En Etiquetas de asignación de costos definidas por el usuario, elija Activar.

Después de crear y activar las etiquetas, AWS genera un informe de asignación de costos con el uso y los costos agrupados según las etiquetas activas. El informe de asignación de costes incluye todos los costes de AWS para cada periodo de facturación. El informe incluye tanto recursos etiquetados como sin etiquetar, para que pueda organizar con claridad los cargos de los recursos.

> **Note**
>
> En la actualidad, los datos transferidos desde DynamoDB no se desglosan según las etiquetas en los informes de asignación de costos.

Para obtener más información, consulte Uso de etiquetas de asignación de costos.

Tablas globales: multiactivas, replicación en varias regiones

Las tablas globales de Amazon DynamoDB son una característica de base de datos totalmente administrada, de varias regiones y multiactiva que proporciona una replicación de datos fácil de usar y un rendimiento de lectura y escritura rápido y local para aplicaciones escaladas de forma global.

Las tablas globales replican automáticamente los datos de las tablas de DynamoDB en todas las Regiones de AWS sin necesidad de que cree y mantenga su propia solución de replicación. Cuando elige las Regiones de AWS para las réplicas de las tablas, las tablas globales gestionan toda la replicación automáticamente. Las tablas globales están disponibles en todas las regiones donde DynamoDB está disponible.

Las tablas globales proporcionan los siguientes beneficios:

- Replicación de forma automática de los datos de las tablas de DynamoDB en las regiones de AWS que elija para ubicar los datos más cerca de los usuarios
- Habilitación de una mayor disponibilidad de las aplicaciones durante el aislamiento o la degradación regional
- Eliminación de la resolución de conflictos de actualización para centrarse en la lógica empresarial de la aplicación

Las tablas globales son ideales para aplicaciones que requieren continuidad empresarial y alta disponibilidad a través de una implementación en varias regiones. Cualquier réplica de una tabla global puede servir para lecturas y escrituras. Las aplicaciones pueden lograr una alta resiliencia con un objetivo de punto de recuperación (RPO) bajo o nulo al desplazar el tráfico a una región diferente si el procesamiento de la aplicación se interrumpe en una región.

Puede configurar una tabla global mediante la consola de administración de AWS. Las tablas globales utilizan las API de DynamoDB existentes para leer y escribir datos en las tablas, por lo que no se requieren cambios en la aplicación. Solo paga por los recursos que aprovisione o use, sin costos iniciales ni compromisos. Para empezar a usar tablas globales, consulte Tutoriales: creación de tablas globales.

Temas

- Funcionamiento de las tablas globales de DynamoDB
- Tutoriales: creación de tablas globales
- Seguridad de las tablas globales de DynamoDB
- Descripción de la facturación de tablas globales en Amazon DynamoDB
- Versiones de tablas globales de DynamoDB
- Prácticas recomendadas para tablas globales

Funcionamiento de las tablas globales de DynamoDB

En las siguientes secciones, se describen los conceptos y el comportamiento de las tablas globales en Amazon DynamoDB.

Conceptos

Las tablas globales son una característica de DynamoDB que replica los datos de las tablas en todas las regiones de AWS.

Una tabla de réplica (o réplica) es una tabla de DynamoDB que funciona como parte de una tabla global. Una tabla global consta de dos o más tablas de réplica en diferentes regiones de AWS. Cada tabla global solo puede tener una réplica por región de AWS. Todas las réplicas de una tabla global comparten el mismo nombre de tabla, esquema de clave principal y datos de elementos.

Cuando una aplicación escribe datos en una réplica en una región, DynamoDB replica automáticamente la escritura en el resto de las réplicas en la tabla global. Para obtener más información sobre cómo empezar a usar tablas globales, consulte Tutoriales: creación de tablas globales.

Versiones

Hay disponibles dos versiones de las tablas globales de DynamoDB: versión 2019.11.21 (actual) y versión 2017.11.29 (antigua). Debe utilizar la versión 2019.11.21 (actual) siempre que sea posible. La información de esta sección de la documentación corresponde a la versión 2019.11.21 (actual). Para obtener más información, consulte Determinación de la versión de una tabla global.

Disponibilidad

Las tablas globales ayudan a mejorar la continuidad empresarial al facilitar la implementación de una arquitectura de alta disponibilidad de varias regiones. Si una carga de trabajo en una única región de AWS se ve afectada, puede cambiar el tráfico de la aplicación a otra región y realizar las operaciones de lectura y escritura en una tabla de réplica diferente en la misma tabla global.

Cada tabla réplica en una tabla global proporciona la misma durabilidad y disponibilidad que una tabla de DynamoDB de una sola región. Las tablas globales ofrecen un Acuerdo de nivel de servicio (SLA) con una disponibilidad del 99,999 %, en comparación con el 99,99 % de las tablas de una sola región.

Modos de coherencia

Al crear una tabla global, puede configurar el modo de consistencia. Las tablas globales admiten dos modos de coherencia: coherencia final de varias regiones (MREC) y coherencia alta de varias regiones (MRSC).

Si no especifica un modo de coherencia al crear una tabla global, esta adoptará de forma predeterminada la coherencia final de varias regiones (MREC). Una tabla global no puede contener réplicas configuradas con diferentes modos de coherencia. No puede cambiar el modo de coherencia de una tabla global después de la creación.

Consistencia posterior de varias regiones (MREC)

La coherencia final de varias regiones (MREC) es el modo de coherencia predeterminado para las tablas globales. Los cambios de elementos en una réplica de tabla global de MREC se replican de forma asíncrona en todas las demás réplicas, normalmente en un segundo o menos. En el improbable caso de que una réplica de una tabla global de MREC quede aislada o se vea afectada, cualquier dato que aún no se haya replicado en otras regiones se replicará cuando la réplica recupere el estado correcto.

Si el mismo elemento se modifica simultáneamente en varias regiones, DynamoDB resolverá el conflicto mediante la modificación con la última marca temporal interna por elemento, lo que se denomina método de resolución de conflictos “el último escritor gana”. Al final, un elemento convergerá en todas las réplicas a la versión creada por la última escritura.

Las operaciones de lectura altamente coherente devuelven la versión más reciente de un elemento si este se ha actualizado por última vez en la región en la que se ha producido la lectura, pero podrían devolver datos obsoletos si el elemento se ha actualizado por última vez en una región diferente. Las escrituras condicionales evalúan la expresión de la condición comparándola con la versión del elemento en la región.

Para crear una tabla global de MREC, se agrega una réplica a una tabla de DynamoDB existente. La agregación de una réplica no tiene impacto en el rendimiento de las tablas de DynamoDB de una sola región ni de las réplicas de tablas globales existentes. Puede agregar réplicas a una tabla global de MREC para ampliar el número de regiones en las que se replican los datos o eliminar réplicas de una tabla global de MREC si ya no son necesarias. Una tabla global de MREC puede tener una réplica en cualquier región en la que DynamoDB esté disponible y puede tener tantas réplicas como regiones haya en la partición de AWS.

Coherencia alta de varias regiones (MRSC)

Puede configurar el modo de coherencia alta de varias regiones (MRSC) al crear una tabla global. Los cambios de elementos en la réplica de una tabla global de MRSC se replican de forma sincrónica en al menos otra región antes de que la operación de escritura devuelva una respuesta correcta. Al convertir una tabla de una sola región existente en una tabla global de MRSC, debe asegurarse de

que la tabla esté vacía hasta que se complete la conversión para garantizar una configuración de inicialización y replicación adecuada.

Las operaciones de lectura altamente coherente en cualquier réplica de MRSC siempre devuelven la versión más reciente de un elemento. Las escrituras condicionales siempre evalúan la expresión de la condición comparándola con la versión más reciente de un elemento.

Una operación de escritura produce un error con una `ReplicatedWriteConflictException` cuando intenta modificar un elemento que ya se está modificando en otra región. Las escrituras que tengan un error con la `ReplicatedWriteConflictException` se pueden reintentar y se realizarán correctamente si el elemento ya no se está modificando en otra región.

Puede configurar una tabla global de MRSC con tres réplicas o con dos réplicas y un testigo. Un testigo es un componente de una tabla global de MRSC que contiene datos escritos en réplicas de tablas globales y proporciona una alternativa opcional a una réplica completa, al tiempo que admite la arquitectura de disponibilidad de MRSC. No puede realizar operaciones de lectura o escritura en un testigo. Un testigo se encuentra en una región diferente a las dos réplicas.

Al crear una tabla global de MRSC, debe elegir las regiones para las réplicas y para la implementación de testigos en el momento de crear la tabla de MRSC. Puede determinar si una tabla global de MRSC tiene un testigo configurado y en qué región, a partir de la salida de la API de `DescribeTable`. El testigo es propiedad y lo administra DynamoDB, y no aparecerá en la cuenta de AWS en la región en la que esté configurado.

Una tabla global de MRSC se debe implementar exactamente en tres regiones. Se crea una tabla global de MRSC mediante la agregación de una réplica y un testigo o dos réplicas a una tabla de DynamoDB existente que no contenga datos. No puede agregar réplicas adicionales a una tabla global de MRSC existente. No puede eliminar una réplica única o un testigo de una tabla global de MRSC. Puede eliminar dos réplicas o eliminar una réplica y un testigo de una tabla global de MRSC, convirtiendo la réplica restante en una tabla de DynamoDB de una sola región.

Las siguientes consideraciones se aplican a las tablas globales de MRSC:

- Al convertir una tabla de una sola región en una tabla global de MRSC, debe asegurarse de que la tabla esté vacía. No se admite la conversión de una tabla de una sola región en una tabla global de MRSC con los elementos existentes. Asegúrese de que no se escriba ningún dato en la tabla durante el proceso de conversión.
- Las tablas globales de MRSC están disponibles en los siguientes conjuntos de regiones:

  - Conjunto de regiones de EE. UU.: Este de EE. UU. (Norte de Virginia), Este de EE. UU. (Ohio), Oeste de EE. UU. (Oregón)
  - Conjunto de regiones de la Unión Europea: Europa (Irlanda), Europa (Londres), Europa (París), Europa (Fráncfort)
  - Conjunto de regiones Asia-Pacífico: Asia-Pacífico (Tokio), Asia-Pacífico (Seúl) y Asia-Pacífico (Osaka).
- Las tablas globales de MRSC no pueden abarcar conjuntos de regiones (por ejemplo, una tabla global de MRSC no puede contener réplicas de conjuntos de regiones de EE. UU. y de la UE).
- El Tiempo de vida (TTL) no se admite para las tablas globales de MRSC.
- Los índices secundarios locales (LSI) no se admiten para las tablas globales de MRSC.
- La información de CloudWatch Contributor Insights solo se comunica para la región en la que se ha producido una operación.

Elección de un modo de coherencia

El criterio clave para elegir un modo de coherencia de varias regiones es si la aplicación prioriza las escrituras de menor latencia y las lecturas altamente coherentes, o si prioriza la coherencia alta global.

Las tablas globales de MREC tendrán latencias de escritura y lectura altamente coherente inferiores en comparación con las tablas globales de MRSC. Las tablas globales de MREC pueden admitir un objetivo de punto de recuperación (RPO) igual al retraso de replicación entre réplicas, normalmente unos segundos, en función de las regiones de las réplicas.

Debe utilizar el modo MREC cuando:

- La aplicación puede tolerar que se devuelvan datos obsoletos de operaciones de lectura altamente coherentes si esos datos se han actualizado en otra región.
- Prioriza las latencias de escritura y de lectura altamente coherente más bajas sobre la consistencia de lectura de varias regiones.
- La estrategia de alta disponibilidad de varias regiones puede tolerar un RPO superior a cero.

Las tablas globales de MRSC tendrán latencias de escritura y de lectura altamente coherente más elevadas que las tablas globales de MREC. Las tablas globales de MRSC admiten un objetivo de punto de recuperación (RPO) de cero.

Debe utilizar el modo MRSC cuando:

- Necesita lecturas altamente coherentes en varias regiones.
- Dé prioridad a la consistencia de lectura global sobre una latencia de escritura menor.
- La estrategia de alta disponibilidad de varias regiones requiera un RPO de cero.

Monitoreo de tablas globales

Las tablas globales configuradas para una coherencia final de varias regiones (MREC) publican la métrica `ReplicationLatency` en CloudWatch. Esta métrica realiza un seguimiento del tiempo transcurrido entre el momento en que un elemento se escribe en una tabla de réplica y el momento en que dicho elemento aparece en otra réplica de la tabla global. `ReplicationLatency` se expresa en milisegundos y se emite para cada par de región de origen y destino en una tabla global.

Los valores típicos de `ReplicationLatency` dependen de la distancia entre las regiones de AWS elegidas, así como de otras variables, como el tipo de carga de trabajo y el rendimiento. Por ejemplo, una réplica de origen en la región Oeste de EE. UU. (Norte de California) (us-west-1) tiene un nivel más bajo de `ReplicationLatency` hacia la región Oeste de EE. UU. (Oregón) (us-west-2) en comparación con la región África (Ciudad del Cabo) (af-south-1).

Un valor en aumento de `ReplicationLatency` podría indicar que las actualizaciones de una réplica no se están propagando hacia otras tablas de réplica de manera puntual. En este caso, puede redirigir temporalmente la actividad de lectura y escritura de la aplicación a otra región de AWS.

Las tablas globales configuradas para coherencia alta de varias regiones (MRSC) no publican una métrica de `ReplicationLatency`.

Pruebas de inyección de errores

Las tablas globales de MREC se integran con el Servicio de inyección de errores de AWS (AWS FIS) para realizar experimentos de inyección de errores en las cargas de trabajo de las tablas globales. Esto le permite probar la respuesta de la aplicación a un aislamiento de región simulado al pausar la replicación hacia y desde una réplica seleccionada. Para obtener más información, consulte Pausar la replicación de una tabla global.

Tiempo de vida (TTL)

Las tablas globales configuradas para MREC admiten la configuración de la eliminación de Tiempo de vida (TTL). La configuración de TTL se sincroniza automáticamente para todas las réplicas en una tabla global. Cuando TTL elimina un elemento de una réplica en una región, la eliminación se replica en todas las demás réplicas de la tabla global. El TTL no consume capacidad de escritura, por lo que no se le cobrará por la eliminación de TTL en la región donde se realizó la eliminación. Sin embargo, se le cobrará por la eliminación replicada en cada una de las demás regiones con una réplica en la tabla global.

La replicación de eliminación de TTL consume la capacidad de escritura de las réplicas en las que se está replicando la eliminación. Las réplicas configuradas para la capacidad aprovisionada pueden limitar las solicitudes si la combinación del rendimiento de escritura y el rendimiento de eliminación de TTL es superior a la capacidad de escritura aprovisionada.

Las tablas globales configuradas para la coherencia alta de varias regiones (MRSC) no admiten la configuración de la eliminación de Tiempo de vida (TTL).

Transmisión

Las tablas globales configuradas para una coherencia final de varias regiones (MREC) replican los cambios leyendo esos cambios desde un Flujo de DynamoDB en una tabla de réplica y aplicándolos a todas las demás tablas de réplica. Por lo tanto, los flujos están habilitados de forma predeterminada en todas las réplicas de una tabla global de MREC y no se pueden desactivar en esas réplicas. El proceso de replicación de MREC puede combinar varios cambios en un corto periodo de tiempo en una sola escritura replicada, lo que hace que el flujo de cada réplica contenga registros ligeramente diferentes. Los registros de flujos en las réplicas de MREC siempre se ordenan por elemento, pero el orden entre los elementos puede variar entre las réplicas.

Las tablas globales configuradas para una coherencia sólida de varias regiones (MRSC) no utilizan los flujos de DynamoDB para la replicación, por lo que los flujos no están habilitados de forma predeterminada en las réplicas de MRSC. Puede habilitar flujos en una réplica de MRSC. Los registros de flujos en las réplicas de MRSC son idénticos para todas las réplicas, incluido el orden de los registros de flujos.

Si desea escribir una aplicación que procese los registros de flujos para cambios que se produjeron en una región específica pero no en otras regiones de una tabla global, puede agregar un atributo a cada elemento que defina en qué región se produjo el cambio para ese elemento. Puede usar este atributo para filtrar los registros de flujos en busca de cambios que se hayan producido en otras

regiones, incluido el uso de filtros de eventos de Lambda para invocar solo las funciones de Lambda para los cambios en una región específica.

Transacciones

En una tabla global configurada para MREC, las operaciones de transacción de DynamoDB (`TransactWriteItems` y `TransactGetItems`) son solo atómicas dentro de la región donde se invocó la operación. Las escrituras transaccionales no se replican como una unidad en todas las regiones, lo que significa que solo algunas de las escrituras de una transacción pueden devolverse mediante operaciones de lectura en otras réplicas en un momento dado.

Por ejemplo, si tiene una tabla global con réplicas en las regiones Este de EE. UU. (Ohio) y Oeste de EE. UU. (Oregón) y realiza una operación `TransactWriteItems` en la región Este de EE. UU. (Ohio), puede observar transacciones completadas parcialmente en la región Oeste de EE. UU. (Oregón) a medida que los cambios se replican. Solo se replicarán los cambios en otras regiones cuando se hayan confirmado en la región de origen.

Las tablas globales configuradas para lograr una coherencia alta de varias regiones (MRSC) no admiten operaciones de transacción y devolverán un error si esas operaciones se invocan en una réplica de MRSC.

Rendimiento de lectura y escritura

La replicación consume capacidad de escritura. Las réplicas configuradas para la capacidad aprovisionada pueden limitar las solicitudes si la combinación del rendimiento de escritura y el rendimiento de replicación es superior a la capacidad de escritura aprovisionada. La capacidad de escritura en el modo bajo demanda se sincroniza para todas las réplicas en una tabla global. Las tablas globales configuradas para la capacidad aprovisionada sincronizan la configuración de escalado automático entre réplicas. La configuración de la capacidad de escritura aprovisionada real puede variar entre las réplicas en función del rendimiento de escritura consumido.

Puede configurar de forma independiente los ajustes de capacidad de lectura para cada réplica en una tabla global. Al agregar una réplica a una tabla global, la capacidad de lectura de la tabla o réplica de origen se utiliza como valor inicial, a menos que se especifique un valor de invalidación.

Sincronización de ajustes

Los ajustes de las tablas globales de DynamoDB son parámetros de configuración que controlan varios aspectos del comportamiento y la replicación de las tablas. Estos ajustes se administran mediante las API del plano de control de DynamoDB y se pueden configurar al crear o modificar

tablas globales. Las tablas globales sincronizan automáticamente determinados ajustes en todas las réplicas para mantener la coherencia, al tiempo que permiten flexibilidad para optimizaciones específicas de cada región. Entender qué ajustes se sincronizan y cómo se comportan le ayuda a configurar la tabla global de manera efectiva. Los ajustes se dividen en tres categorías principales en función de cómo se sincronizan entre las réplicas.

Los siguientes ajustes siempre se sincronizan entre las réplicas de una tabla global:

- Modo de capacidad (capacidad aprovisionada o bajo demanda)
- Capacidad de escritura aprovisionada de tabla
- Escalado automático de la escritura de tablas
- Definición del índice secundario global (GSI)
- Capacidad de escritura aprovisionada por GSI
- Escalado automático de escritura de GSI
- Tipo de cifrado del servidor (SSE)
- Definición de flujos en modo de MREC
- Tiempo de vida (TTL)
- Rendimiento en caliente
- Rendimiento de escritura máximo bajo demanda

Los siguientes ajustes se sincronizan entre las réplicas, pero se pueden invalidar por réplica:

- La capacidad de lectura aprovisionada de la tabla
- Escalado automático de lectura de tablas
- Capacidad de lectura aprovisionada de GSI
- Escalado automático de lectura de GSI
- Clase de tabla
- Rendimiento de lectura máximo bajo demanda

> **Note**
>
> Los valores de configuración anulables se modifican si la configuración se cambia en cualquier otra réplica. Como ejemplo, tiene una tabla global de MREC con réplicas en Este de EE. UU. (Norte de Virginia) y Oeste de EE. UU. (Oregón). La réplica de Este de EE. UU.

(Norte de Virginia) tiene un rendimiento de lectura aprovisionado establecido en 200 RCU. La réplica en Oeste de EE. UU. (Oregón) tiene una invalidación del rendimiento de lectura aprovisionada establecida en 100 RCU. Si actualiza la configuración de rendimiento de lectura aprovisionada en la réplica de Este de EE. UU. (Norte de Virginia) de 200 a 300 RCU, el nuevo valor de rendimiento de lectura aprovisionado también se aplicará a la réplica en Oeste de EE. UU. (Oregón). Esto cambia la configuración de rendimiento de lectura aprovisionada para la réplica de Oeste de EE. UU. (Oregón) del valor invalidado de 100 RCU al nuevo valor de 300 RCU.

Los siguientes ajustes nunca se sincronizan entre réplicas:

- Protección contra eliminación
- Recuperación en un momento dado
- Etiquetas
- Habilitación de CloudWatch Contributor Insights para tablas
- Habilitación de CloudWatch Contributor Insights para GSI
- Definición de Kinesis Data Streams
- Políticas de recursos
- Definición de flujos en modo de MRSC

Todos los demás ajustes no están sincronizados entre réplicas.

DynamoDB Accelerator (DAX)

Las réplicas de escrituras en tablas globales omiten DynamoDB Accelerator (DAX) y actualizan DynamoDB directamente. Como resultado, las cachés de DAX pueden quedar obsoletas, ya que las escrituras no actualizan la caché de DAX. Las cachés de DAX configuradas para réplicas de tablas globales solo se actualizarán cuando caduque el TTL de la caché.

Consideraciones para la administración de tablas globales

No puede eliminar una tabla utilizada para agregar una nueva réplica de tabla global hasta que hayan transcurrido 24 horas desde que se creó la nueva réplica.

Si desactiva una región de AWS que contiene réplicas de tablas globales, esas réplicas se convierten permanentemente en tablas de una sola región 20 horas después de desactivar la región.

Tutoriales: creación de tablas globales

Esta sección proporciona instrucciones paso a paso para crear tablas globales de DynamoDB configuradas para su modo de coherencia preferido. Elija los modos de coherencia final de varias regiones (MREC) o coherencia alta de varias regiones (MRSC) según los requisitos de la aplicación.

Las tablas globales de MREC proporcionan una latencia de escritura más baja con coherencia final en todas las Regiones de AWS. Las tablas globales de MRSC proporcionan lecturas altamente coherentes en todas las regiones, con latencias de escritura ligeramente superiores a las de MREC. Elija el modo de coherencia que mejor se adapte a las necesidades de la aplicación en cuanto a coherencia de datos, latencia y disponibilidad.

Temas

- Creación de una tabla global configurada para MREC
- Creación de una tabla global configurada para MRSC

Creación de una tabla global configurada para MREC

En esta sección se muestra cómo crear una tabla global con el modo de coherencia final de varias regiones (MREC). MREC es el modo de coherencia predeterminado para tablas globales y proporciona escrituras de baja latencia con replicación asincrónica en todas las Regiones de AWS. Los cambios realizados en un elemento de una región suelen replicarse en todas las demás regiones en un segundo. Esto hace que MREC sea ideal para aplicaciones que priorizan una baja latencia de escritura y pueden tolerar breves periodos en los que diferentes regiones pueden devolver versiones de datos ligeramente diferentes.

Puede crear tablas globales de MREC con réplicas en cualquier región de AWS donde DynamoDB esté disponible y agregar o eliminar réplicas en cualquier momento. Los ejemplos siguientes muestran cómo crear una tabla global de MREC con réplicas en varias regiones.

Creación de una tabla global de MREC mediante la consola de DynamoDB

Siga estos pasos para crear una tabla global mediante la Consola de administración de AWS. En el ejemplo siguiente, se crea una tabla global con tablas de réplica en Estados Unidos y Europa.

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. Para este ejemplo, elija Este de EE. UU. (Ohio) en el selector de regiones en la barra de navegación.

3. En el panel de navegación del lado izquierdo de la consola, elija Tablas.
4. Seleccione Create Table (Crear tabla).
5. En la página Crear tabla:
   a. En Nombre de la tabla, introduzca **`Music`**.
   b. En Partition key (Clave de partición), ingrese **`Artist`**.
   c. Para Clave de clasificación, ingrese **`SongTitle`**.
   d. Deje los demás valores predeterminados y elija Crear tabla.

      Esta nueva tabla le servirá de primera tabla de réplica en una nueva tabla global. Será el prototipo para crear otras tablas de réplica que quiera añadir más tarde.
6. Cuando la tabla se active:
   a. Elija la tabla Música en la lista de tablas.
   b. Elija la pestaña Tablas globales.
   c. Elija Create replica (Crear réplica).
7. En el menú desplegable Regiones de replicación disponibles, elija Oeste de EE. UU. (Oregón) us-west-2.

   La consola asegura que no haya ninguna tabla con el mismo nombre en la región seleccionada. Si ya existe una tabla con el mismo nombre, debe eliminar la tabla existente antes de crear una nueva tabla de réplica en esa región.
8. Elija Create replica (Crear réplica). Esto comienza el proceso de creación de la tabla en la región Oeste de EE. UU (Oregón) us-west-2.

   La pestaña Tablas globales de la tabla Música (y de cualquier otra tabla de réplica) indica que la tabla se ha replicado en varias regiones.
9. Agregue otra región repitiendo los pasos anteriores, pero elija Europa (Fráncfort) eu-central-1 como región.
10. Para probar la replicación:
    a. Asegúrese de que utiliza la Consola de administración de AWS en la región Este de EE. UU. (Ohio).
    b. Elija Explorar elementos de la tabla.
    c. Seleccione Crear elemento.

d. Ingrese **`item_1`** para Artista y **`Song Value 1`** para Título de la canción.

e. Seleccione Crear elemento.

11. Compruebe la replicación cambiando a las otras regiones:

a. En el selector de regiones en la esquina superior derecha, elija Europa (Fráncfort).

b. Compruebe que la tabla Música contiene el elemento que ha creado.

c. Repita la verificación para Oeste de EE. UU. (Oregón).

Creación de una tabla global de MREC mediante AWS CLI o Java

CLI

En el siguiente ejemplo de código, se muestra cómo administrar las tablas globales de DynamoDB con replicación de varias regiones con coherencia final (MREC).

- Cree una tabla con replicación de varias regiones (MREC).
- Coloque y obtenga elementos de las tablas de réplica.
- Elimine réplicas una por una.
- Efectúe una limpieza eliminando la tabla.

AWS CLI con Bash script

Cree una tabla con replicación de varias regiones.

```
# Step 1: Create a new table (MusicTable) in US East (Ohio), with DynamoDB
 Streams enabled (NEW_AND_OLD_IMAGES)
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2

# Step 2: Create an identical MusicTable table in US East (N. Virginia)
```

```
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2

# Step 3: Create a table in Europe (Ireland)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2
```

Describa la tabla de varias regiones.

```
# Step 4: View the list of replicas created using describe-table
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

Coloque elementos en una tabla de réplica.

```
# Step 5: To verify that replication is working, add a new item to the Music
 table in US East (Ohio)
```

```
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

Obtenga elementos de las tablas de réplica.

```
# Step 6: Wait for a few seconds, and then check to see whether the item has
 been
# successfully replicated to US East (N. Virginia) and Europe (Ireland)
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1

aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

Elimine réplicas.

```
# Step 7: Delete the replica table in Europe (Ireland) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2

# Delete the replica table in US East (N. Virginia) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
```

```
    {
      "Delete": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2
```

Efectúe una limpieza eliminando la tabla.

```
# Clean up: Delete the primary table
aws dynamodb delete-table --table-name MusicTable --region us-east-2

echo "Global table demonstration complete."
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de comandos de AWS CLI.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateTable

Java

En el ejemplo de código siguiente, se muestra cómo crear y administrar las tablas globales de DynamoDB con réplicas en varias regiones.

- Cree una tabla con un índice secundario global y flujos de DynamoDB.
- Agregue réplicas en diferentes regiones para crear una tabla global.
- Elimine réplicas de una tabla global.
- Agregue elementos de prueba para verificar la replicación en todas las regiones.
- Describa la configuración de la tabla global y el estado de las réplicas.

SDK para Java 2.x

Cree una tabla con un índice secundario global y flujos de DynamoDB mediante AWS SDK for Java 2.x.

```
   public static CreateTableResponse createTableWithGSI(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
indexName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (indexName == null || indexName.trim().isEmpty()) {
           throw new IllegalArgumentException("Index name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating table: " + tableName + " with GSI: " +
indexName);

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
               .tableName(tableName)
               .attributeDefinitions(
                   AttributeDefinition.builder()
                       .attributeName("Artist")
                       .attributeType(ScalarAttributeType.S)
                       .build(),
                   AttributeDefinition.builder()
                       .attributeName("SongTitle")
                       .attributeType(ScalarAttributeType.S)
                       .build())
               .keySchema(
                   KeySchemaElement.builder()
                       .attributeName("Artist")
                       .keyType(KeyType.HASH)
                       .build(),
                   KeySchemaElement.builder()
                       .attributeName("SongTitle")
                       .keyType(KeyType.RANGE)
```

```
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .globalSecondaryIndexes(GlobalSecondaryIndex.builder()
                    .indexName(indexName)
                    .keySchema(KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.HASH)
                        .build())
                    .projection(

Projection.builder().projectionType(ProjectionType.ALL).build())
                    .build())
                .streamSpecification(StreamSpecification.builder()
                    .streamEnabled(true)
                    .streamViewType(StreamViewType.NEW_AND_OLD_IMAGES)
                    .build())
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

Espere a que una tabla se active mediante AWS SDK for Java 2.x.

```
    public static void waitForTableActive(final DynamoDbClient dynamoDbClient,
final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
```

```
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Waiting for table to become active: " + tableName);

            try (DynamoDbWaiter waiter =
                DynamoDbWaiter.builder().client(dynamoDbClient).build()) {
                DescribeTableRequest request =
                    DescribeTableRequest.builder().tableName(tableName).build();

                waiter.waitUntilTableExists(request);
                LOGGER.info("Table is now active: " + tableName);
            }

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to wait for table to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

Agregue una réplica para crear o ampliar una tabla global mediante AWS SDK for Java 2.x.

```
    public static UpdateTableResponse addReplica(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final String indexName,
        final Long readCapacity) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
```

```
        if (indexName == null || indexName.trim().isEmpty()) {
            throw new IllegalArgumentException("Index name cannot be null or
empty");
        }
        if (readCapacity == null || readCapacity <= 0) {
            throw new IllegalArgumentException("Read capacity must be a positive
number");
        }

        try {
            LOGGER.info("Adding replica in region: " + replicaRegion.id() + " for
table: " + tableName);

            // Create a ReplicationGroupUpdate for adding a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .create(builder -> builder.regionName(replicaRegion.id())
                    .globalSecondaryIndexes(ReplicaGlobalSecondaryIndex.builder()
                        .indexName(indexName)

.provisionedThroughputOverride(ProvisionedThroughputOverride.builder()
                            .readCapacityUnits(readCapacity)
                            .build())
                        .build())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica addition initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
```

```
    }
```

Elimine una réplica de una tabla global mediante AWS SDK for Java 2.x.

```
    public static UpdateTableResponse removeReplica(
        final DynamoDbClient dynamoDbClient, final String tableName, final Region
replicaRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }

        try {
            LOGGER.info("Removing replica in region: " + replicaRegion.id() + "
for table: " + tableName);

            // Create a ReplicationGroupUpdate for removing a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .delete(builder ->
builder.regionName(replicaRegion.id()).build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica removal initiated in region: " +
replicaRegion.id());

            return response;
```

```
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to remove replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

Agregue elementos de prueba para verificar la replicación mediante AWS SDK for Java 2.x.

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Adding test item to table: " + tableName);

            Map<String,
software.amazon.awssdk.services.dynamodb.model.AttributeValue> item = new
HashMap<>();
            item.put(
                "Artist",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(artist)
                    .build());
            item.put(
                "SongTitle",
```

```
software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(songTitle)
                    .build());

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw e;
        }
    }
```

Describa la configuración de la tabla global y las réplicas mediante AWS SDK for Java 2.x.

```
    public static DescribeTableResponse describeTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Describing table: " + tableName);

            DescribeTableRequest request =
                DescribeTableRequest.builder().tableName(tableName).build();

            DescribeTableResponse response =
dynamoDbClient.describeTable(request);
```

```
            LOGGER.info("Table status: " + response.table().tableStatus());
            if (response.table().replicas() != null
                && !response.table().replicas().isEmpty()) {
                LOGGER.info("Number of replicas: " +
response.table().replicas().size());
                response.table()
                    .replicas()
                    .forEach(replica -> LOGGER.info(
                        "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

Ejemplo completo de operaciones de tablas globales mediante AWS SDK for Java 2.x.

```
    public static void exampleUsage(final Region sourceRegion, final Region
replicaRegion) {

        String tableName = "Music";
        String indexName = "SongTitleIndex";
        Long readCapacity = 15L;

        // Create DynamoDB client for the source region
        try (DynamoDbClient dynamoDbClient =
            DynamoDbClient.builder().region(sourceRegion).build()) {

            try {
                // Step 1: Create the initial table with GSI and streams
                LOGGER.info("Step 1: Creating table in source region: " +
sourceRegion.id());
```

```
                createTableWithGSI(dynamoDbClient, tableName, indexName);

                // Step 2: Wait for table to become active
                LOGGER.info("Step 2: Waiting for table to become active");
                waitForTableActive(dynamoDbClient, tableName);

                // Step 3: Add replica in destination region
                LOGGER.info("Step 3: Adding replica in region: " +
replicaRegion.id());
                addReplica(dynamoDbClient, tableName, replicaRegion, indexName,
readCapacity);

                // Step 4: Wait a moment for replica creation to start
                Thread.sleep(5000);

                // Step 5: Describe table to view replica information
                LOGGER.info("Step 5: Describing table to view replicas");
                describeTable(dynamoDbClient, tableName);

                // Step 6: Add a test item to verify replication
                LOGGER.info("Step 6: Adding test item to verify replication");
                putTestItem(dynamoDbClient, tableName, "TestArtist", "TestSong");

                LOGGER.info("Global table setup completed successfully!");
                LOGGER.info("You can verify replication by checking the item in
region: " + replicaRegion.id());

                // Step 7: Remove replica and clean up table
                LOGGER.info("Step 7: Removing replica from region: " +
replicaRegion.id());
                removeReplica(dynamoDbClient, tableName, replicaRegion);
                DeleteTableResponse deleteTableResponse =
dynamoDbClient.deleteTable(
                    DeleteTableRequest.builder().tableName(tableName).build());
                LOGGER.info("MREC global table demonstration completed
successfully!");

            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                throw new RuntimeException("Thread was interrupted", e);
            } catch (DynamoDbException e) {
                LOGGER.severe("DynamoDB operation failed: " + e.getMessage());
                throw e;
            }
```

```
        }
    }
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for Java 2.x.
  - CreateTable
  - DescribeTable
  - PutItem
  - UpdateTable

Creación de una tabla global configurada para MRSC

En esta sección se muestra cómo crear una tabla global de coherencia alta de varias regiones (MRSC). Las tablas globales de MRSC replican de forma sincrónica los cambios de los elementos en todas las regiones, lo que garantiza que las operaciones de lectura altamente coherente en cualquier réplica siempre devuelvan la versión más reciente de un elemento. Al convertir una tabla de una sola región en una tabla global de MRSC, debe asegurarse de que la tabla esté vacía. No se admite la conversión de una tabla de una sola región en una tabla global de MRSC con los elementos existentes. Asegúrese de que no se escriba ningún dato en la tabla durante el proceso de conversión.

Puede configurar una tabla global de MRSC con tres réplicas o dos réplicas y un testigo. Al crear una tabla global de MRSC, debe elegir las regiones en las que se despliegan las réplicas y un testigo opcional. En el siguiente ejemplo, se crea una tabla global de MRSC con réplicas en las regiones Este de EE. UU. (Norte de Virginia) y Este de EE. UU. (Ohio), con un testigo en la región Oeste de EE. UU. (Oregón).

> **Note**
>
> Antes de crear una tabla global, compruebe que los límites de rendimiento de la cuota de servicio sean coherentes en todas las regiones de destino, ya que es necesario para crear una tabla global. Para obtener más información sobre los límites de rendimiento de las tablas globales, consulte Cuotas de tablas globales.

Creación de una tabla global de MRSC mediante la consola de DynamoDB

Siga estos pasos para crear una tabla global de MRSC mediante la Consola de administración de AWS.

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el selector de regiones de la barra de navegación, elija una región en la que se admiten tablas globales con MRSC, como **`us-east-2`**.
3. En el panel de navegación, elija Tablas.
4. Seleccione Create table (Creación de tabla).
5. En la página Crear tabla:
   a. En Nombre de la tabla, introduzca **`Music`**.
   b. En Clave de partición, ingrese **`Artist`** y mantenga la Cadena predeterminada.
   c. En Clave de clasificación, ingrese **`SongTitle`** y mantenga el tipo de Cadena predeterminada.
   d. Deje los demás valores predeterminados y elija Crear tabla

      Esta nueva tabla le servirá de primera tabla de réplica en una nueva tabla global. Será el prototipo para crear otras tablas de réplica que quiera añadir más tarde.
6. Espere a que la tabla se active y luego selecciónela en la lista de tablas.
7. Elija la pestaña Tablas globales y, a continuación, seleccione Crear réplica.
8. En la página Crear réplica:
   a. En Consistencia de varias regiones, elija Consistencia alta.
   b. Para Región de replicación 1, elija **`US East (N. Virginia) us-east-1`**.
   c. Para Región de replicación 2, elija **`US West (Oregon) us-west-2`**.
   d. Compruebe Configurar como testigo para la región Oeste de EE. UU. (Oregón).
   e. Elija Crear réplicas.
9. Espere a que se complete el proceso de creación de la réplica y el testigo. El estado de la réplica se mostrará como Activo cuando la tabla esté lista para usarse.

Creación de una tabla global de MRSC mediante la AWS CLI o Java

Antes de empezar, asegúrese de que la entidad principal de IAM tenga los permisos necesarios para crear una tabla global de MRSC con una región testigo.

La política de IAM de ejemplo siguiente le permite crear una tabla de DynamoDB (`MusicTable`) en Este de EE. UU. (Ohio) con una réplica en Este de EE. UU. (Norte de Virginia) y una región testigo en Oeste de EE. UU. (Oregón):

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:CreateTableReplica",
                "dynamodb:CreateGlobalTableWitness",
                "dynamodb:DescribeTable",
                "dynamodb:UpdateTable",
                "dynamodb:DeleteTable",
                "dynamodb:DeleteTableReplica",
                "dynamodb:DeleteGlobalTableWitness",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:UpdateItem",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/MusicTable",
                "arn:aws:dynamodb:us-east-2:123456789012:table/MusicTable",
                "arn:aws:dynamodb:us-west-2:123456789012:table/MusicTable"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
```

```
            "Resource": "arn:aws:iam::*:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "replication.dynamodb.amazonaws.com"
                }
            }
        }
    ]
}
```

Los ejemplos de código siguientes, muestran cómo crear y administrar las tablas globales de DynamoDB con coherencia alta de varias regiones (MRSC).

- Cree una tabla con coherencia alta de varias regiones.
- Verifique la configuración de MRSC y el estado de la réplica.
- Pruebe la coherencia alta entre regiones con lecturas inmediatas.
- Realice escrituras condicionales con garantías de MRSC.
- Limpie los recursos de la tabla global de MRSC.

Bash

AWS CLI con Bash script

Cree una tabla con coherencia alta de varias regiones.

```
# Step 1: Create a new table in us-east-2 (primary region for MRSC)
# Note: Table must be empty when enabling MRSC
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --region us-east-2

# Wait for table to become active
```

```
aws dynamodb wait table-exists --table-name MusicTable --region us-east-2

# Step 2: Add replica and witness with Multi-Region Strong Consistency
# MRSC requires exactly three replicas in supported regions
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Create": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Create": {"RegionName": "us-west-2"}}]' \
    --multi-region-consistency STRONG \
    --region us-east-2
```

Verifique la configuración de MRSC y el estado de la réplica.

```
# Verify the global table configuration and MRSC setting
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

Pruebe la coherencia alta con lecturas inmediatas entre regiones.

```
# Write an item to the primary region
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"},"Album":
 {"S":"The Beatles 1967-1970"},"Year": {"N":"1968"}}' \
    --region us-east-2

# Read the item from replica region to verify strong consistency (cannot read or
 write to witness)
# No wait time needed - MRSC provides immediate consistency
echo "Reading from us-east-1 (immediate consistency):"
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --consistent-read \
    --region us-east-1
```

Realice escrituras condicionales con garantías de MRSC.

```
# Perform a conditional update from a different region
# This demonstrates that conditions work consistently across all regions
aws dynamodb update-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --update-expression "SET #rating = :rating" \
    --condition-expression "attribute_exists(Artist)" \
    --expression-attribute-names '{"#rating": "Rating"}' \
    --expression-attribute-values '{":rating": {"N":"5"}}' \
    --region us-east-1
```

Limpie los recursos de la tabla global de MRSC.

```
# Remove replica tables (must be done before deleting the primary table)
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Delete": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Delete": {"RegionName": "us-west-2"}}]' \
    --region us-east-2

# Wait for replicas to be deleted
echo "Waiting for replicas to be deleted..."
sleep 30

# Delete the primary table
aws dynamodb delete-table \
    --table-name MusicTable \
    --region us-east-2
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de comandos de AWS CLI.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem

- UpdateItem
- UpdateTable

Java

SDK para Java 2.x

Cree una tabla regional lista para la conversión de MRSC con AWS SDK for Java 2.x.

```
   public static CreateTableResponse createRegionalTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating regional table: " + tableName + " (must be
empty for MRSC)");

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
               .tableName(tableName)
               .attributeDefinitions(
                   AttributeDefinition.builder()
                       .attributeName("Artist")
                       .attributeType(ScalarAttributeType.S)
                       .build(),
                   AttributeDefinition.builder()
                       .attributeName("SongTitle")
                       .attributeType(ScalarAttributeType.S)
                       .build())
               .keySchema(
                   KeySchemaElement.builder()
                       .attributeName("Artist")
                       .keyType(KeyType.HASH)
                       .build(),
                   KeySchemaElement.builder()
                       .attributeName("SongTitle")
```

```
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Regional table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create regional table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to create regional table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Convierta una tabla regional en MRSC utilizando réplicas y testigos con AWS SDK for Java 2.x.

```
    public static UpdateTableResponse convertToMRSCWithWitness(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
```

```
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("Converting table to MRSC with replica in " +
replicaRegion.id() + " and witness in "
                + witnessRegion.id());

            // Create replica update using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .create(CreateReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Create witness update
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .create(CreateGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .multiRegionConsistency(MultiRegionConsistency.STRONG)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC conversion initiated. Status: "
                + response.tableDescription().tableStatus());
            LOGGER.info("UpdateTableResponse full object: " + response);
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to convert table to MRSC: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to convert table to MRSC: " + tableName)
                .cause(e)
```

```
                .build();
        }
    }
```

Describa la configuración de una tabla global de MRSC con AWS SDK for Java 2.x.

```
   public static DescribeTableResponse describeMRSCTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Describing MRSC global table: " + tableName);

           DescribeTableRequest request =
               DescribeTableRequest.builder().tableName(tableName).build();

           DescribeTableResponse response =
dynamoDbClient.describeTable(request);

           LOGGER.info("Table status: " + response.table().tableStatus());
           LOGGER.info("Multi-region consistency: " +
response.table().multiRegionConsistency());

           if (response.table().replicas() != null
               && !response.table().replicas().isEmpty()) {
               LOGGER.info("Number of replicas: " +
response.table().replicas().size());
               response.table()
                   .replicas()
                   .forEach(replica -> LOGGER.info(
                       "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
           }

           if (response.table().globalTableWitnesses() != null
```

```
                && !response.table().globalTableWitnesses().isEmpty()) {
                LOGGER.info("Number of witnesses: "
                    + response.table().globalTableWitnesses().size());
                response.table()
                    .globalTableWitnesses()
                    .forEach(witness -> LOGGER.info(
                        "Witness region: " + witness.regionName() + ", Status: "
+ witness.witnessStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Table not found: " + tableName)
                .cause(e)
                .build();
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to describe table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Agregue elementos de prueba para verificar la coherencia alta de MRSC con AWS SDK for Java 2.x.

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String album,
        final String year) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
```

```
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Adding test item to MRSC global table: " + tableName);

            Map<String, AttributeValue> item = new HashMap<>();
            item.put("Artist", AttributeValue.builder().s(artist).build());
            item.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            if (album != null && !album.trim().isEmpty()) {
                item.put("Album", AttributeValue.builder().s(album).build());
            }
            if (year != null && !year.trim().isEmpty()) {
                item.put("Year", AttributeValue.builder().n(year).build());
            }

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully with strong consistency");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to add test item to table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Lea los elementos con lecturas coherentes de las réplicas de MRSC con AWS SDK for Java 2.x.

```
   public static GetItemResponse getItemWithConsistentRead(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (artist == null || artist.trim().isEmpty()) {
           throw new IllegalArgumentException("Artist cannot be null or empty");
       }
       if (songTitle == null || songTitle.trim().isEmpty()) {
           throw new IllegalArgumentException("Song title cannot be null or
empty");
       }

       try {
           LOGGER.info("Reading item from MRSC global table with consistent
read: " + tableName);

           Map<String, AttributeValue> key = new HashMap<>();
           key.put("Artist", AttributeValue.builder().s(artist).build());
           key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

           GetItemRequest getItemRequest = GetItemRequest.builder()
               .tableName(tableName)
               .key(key)
               .consistentRead(true)
               .build();

           GetItemResponse response = dynamoDbClient.getItem(getItemRequest);

           if (response.hasItem()) {
               LOGGER.info("Item found with strong consistency - no wait time
needed");
```

```
            } else {
                LOGGER.info("Item not found");
            }

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to read item from table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to read item from table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Realice actualizaciones condicionales con garantías de MRSC mediante AWS SDK for Java 2.x.

```
    public static UpdateItemResponse performConditionalUpdate(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String rating) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }
        if (rating == null || rating.trim().isEmpty()) {
            throw new IllegalArgumentException("Rating cannot be null or empty");
```

```
        }

        try {
            LOGGER.info("Performing conditional update on MRSC global table: " +
tableName);

            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            Map<String, String> expressionAttributeNames = new HashMap<>();
            expressionAttributeNames.put("#rating", "Rating");

            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
            expressionAttributeValues.put(
                ":rating", AttributeValue.builder().n(rating).build());

            UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #rating = :rating")
                .conditionExpression("attribute_exists(Artist)")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();

            UpdateItemResponse response =
dynamoDbClient.updateItem(updateItemRequest);
            LOGGER.info("Conditional update successful - demonstrates strong
consistency");

            return response;

        } catch (ConditionalCheckFailedException e) {
            LOGGER.warning("Conditional check failed: " + e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to perform conditional update: " + tableName +
" - " + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to perform conditional update: " + tableName)
                .cause(e)
                .build();
```

```
        }
    }
```

Espere a que las réplicas de MRSC y los testigos se activen mediante AWS SDK for Java 2.x.

```
    public static void waitForMRSCReplicasActive(
        final DynamoDbClient dynamoDbClient, final String tableName, final int
maxWaitTimeSeconds)
        throws InterruptedException {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (maxWaitTimeSeconds <= 0) {
            throw new IllegalArgumentException("Max wait time must be positive");
        }

        try {
            LOGGER.info("Waiting for MRSC replicas and witnesses to become
active: " + tableName);

            final long startTime = System.currentTimeMillis();
            final long maxWaitTimeMillis = maxWaitTimeSeconds * 1000L;
            int backoffSeconds = 5; // Start with 5 second intervals
            final int maxBackoffSeconds = 30; // Cap at 30 seconds

            while (System.currentTimeMillis() - startTime < maxWaitTimeMillis) {
                DescribeTableResponse response =
describeMRSCTable(dynamoDbClient, tableName);

                boolean allActive = true;
                StringBuilder statusReport = new StringBuilder();

                if (response.table().multiRegionConsistency() == null
                    || !MultiRegionConsistency.STRONG
                        .toString()

.equals(response.table().multiRegionConsistency().toString())) {
```

```
                    allActive = false;
                    statusReport
                        .append("MultiRegionConsistency: ")
                        .append(response.table().multiRegionConsistency())
                        .append(" ");
                }
                if (response.table().replicas() == null
                    || response.table().replicas().isEmpty()) {
                    allActive = false;
                    statusReport.append("No replicas found. ");
                }
                if (response.table().globalTableWitnesses() == null
                    || response.table().globalTableWitnesses().isEmpty()) {
                    allActive = false;
                    statusReport.append("No witnesses found. ");
                }

                // Check table status
                if (!"ACTIVE".equals(response.table().tableStatus().toString()))
{
                    allActive = false;
                    statusReport
                        .append("Table: ")
                        .append(response.table().tableStatus())
                        .append(" ");
                }

                // Check replica status
                if (response.table().replicas() != null) {
                    for (var replica : response.table().replicas()) {
                        if (!"ACTIVE".equals(replica.replicaStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Replica(")
                                .append(replica.regionName())
                                .append("): ")
                                .append(replica.replicaStatus())
                                .append(" ");
                        }
                    }
                }

                // Check witness status
```

```
                if (response.table().globalTableWitnesses() != null) {
                    for (var witness : response.table().globalTableWitnesses()) {
                        if (!"ACTIVE".equals(witness.witnessStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Witness(")
                                .append(witness.regionName())
                                .append("): ")
                                .append(witness.witnessStatus())
                                .append(" ");
                        }
                    }
                }

                if (allActive) {
                    LOGGER.info("All MRSC replicas and witnesses are now active:
" + tableName);
                    return;
                }

                LOGGER.info("Waiting for MRSC components to become active.
Status: " + statusReport.toString());
                LOGGER.info("Next check in " + backoffSeconds + " seconds...");

                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, maxBackoffSeconds);
            }

            throw DynamoDbException.builder()
                .message("Timeout waiting for MRSC replicas to become active
after " + maxWaitTimeSeconds + " seconds")
                .build();

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("Failed to wait for MRSC replicas to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

Limpie las réplicas y los testigos de MRSC mediante AWS SDK for Java 2.x.

```
   public static UpdateTableResponse cleanupMRSCReplicas(
       final DynamoDbClient dynamoDbClient,
       final String tableName,
       final Region replicaRegion,
       final Region witnessRegion) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (replicaRegion == null) {
           throw new IllegalArgumentException("Replica region cannot be null");
       }
       if (witnessRegion == null) {
           throw new IllegalArgumentException("Witness region cannot be null");
       }

       try {
           LOGGER.info("Cleaning up MRSC replicas and witnesses for table: " +
tableName);

           // Remove replica using ReplicationGroupUpdate
           ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
               .delete(DeleteReplicationGroupMemberAction.builder()
                   .regionName(replicaRegion.id())
                   .build())
               .build();

           // Remove witness
           GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
               .delete(DeleteGlobalTableWitnessGroupMemberAction.builder()
                   .regionName(witnessRegion.id())
                   .build())
               .build();

           UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
               .tableName(tableName)
```

```
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC cleanup initiated - removing replica and witness.
Response: " + response);

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to cleanup MRSC replicas: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to cleanup MRSC replicas: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Complete la demostración del flujo de trabajo de MRSC mediante AWS SDK for Java 2.x.

```
    public static void demonstrateCompleteMRSCWorkflow(
        final DynamoDbClient primaryClient,
        final DynamoDbClient replicaClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion)
        throws InterruptedException {

        if (primaryClient == null) {
            throw new IllegalArgumentException("Primary DynamoDB client cannot be
null");
        }
        if (replicaClient == null) {
            throw new IllegalArgumentException("Replica DynamoDB client cannot be
null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
```

```
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("=== Starting Complete MRSC Workflow Demonstration ===");

            // Step 1: Create an empty single-Region table
            LOGGER.info("Step 1: Creating empty single-Region table");
            createRegionalTable(primaryClient, tableName);

            // Use the existing GlobalTableOperations method for basic table
waiting
            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 2: Convert to MRSC with replica and witness
            LOGGER.info("Step 2: Converting to MRSC with replica and witness");
            convertToMRSCWithWitness(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for MRSC conversion to complete using MRSC-specific waiter
            LOGGER.info("Waiting for MRSC conversion to complete...");
            waitForMRSCReplicasActive(primaryClient, tableName);

            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 3: Verify MRSC configuration
            LOGGER.info("Step 3: Verifying MRSC configuration");
            describeMRSCTable(primaryClient, tableName);

            // Step 4: Test strong consistency with data operations
            LOGGER.info("Step 4: Testing strong consistency with data
operations");

            // Add test item to primary region
```

```
            putTestItem(primaryClient, tableName, "The Beatles", "Hey Jude", "The
Beatles 1967-1970", "1968");

            // Immediately read from replica region (no wait needed with MRSC)
            LOGGER.info("Reading from replica region immediately (strong
consistency):");
            GetItemResponse getResponse =
                getItemWithConsistentRead(replicaClient, tableName, "The
Beatles", "Hey Jude");

            if (getResponse.hasItem()) {
                LOGGER.info("# Strong consistency verified - item immediately
available in replica region");
            } else {
                LOGGER.warning("# Item not found in replica region");
            }

            // Test conditional update from replica region
            LOGGER.info("Testing conditional update from replica region:");
            performConditionalUpdate(replicaClient, tableName, "The Beatles",
"Hey Jude", "5");
            LOGGER.info("# Conditional update successful - demonstrates strong
consistency");

            // Step 5: Cleanup
            LOGGER.info("Step 5: Cleaning up resources");
            cleanupMRSCReplicas(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for cleanup to complete using basic table waiter
            LOGGER.info("Waiting for replica cleanup to complete...");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // "Halt" until replica/witness cleanup is complete
            DescribeTableResponse cleanupVerification =
describeMRSCTable(primaryClient, tableName);
            int backoffSeconds = 5; // Start with 5 second intervals
            while (cleanupVerification.table().multiRegionConsistency() != null)
{
                LOGGER.info("Waiting additional time (" + backoffSeconds + "
seconds) for MRSC cleanup to complete...");
                tempWait(backoffSeconds);

                // Exponential backoff with cap
```

```
                backoffSeconds = Math.min(backoffSeconds * 2, 30);
                cleanupVerification = describeMRSCTable(primaryClient,
tableName);
            }

            // Delete the primary table
            deleteTable(primaryClient, tableName);

            LOGGER.info("=== MRSC Workflow Demonstration Complete ===");
            LOGGER.info("");
            LOGGER.info("Key benefits of Multi-Region Strong Consistency
(MRSC):");
            LOGGER.info("- Immediate consistency across all regions (no eventual
consistency delays)");
            LOGGER.info("- Simplified application logic (no need to handle
eventual consistency)");
            LOGGER.info("- Support for conditional writes and transactions across
regions");
            LOGGER.info("- Consistent read operations from any region without
waiting");

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("MRSC workflow failed: " + e.getMessage());
            throw e;
        }
    }
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for Java 2.x.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Seguridad de las tablas globales de DynamoDB

Las réplicas de tablas globales son tablas de DynamoDB, por lo que se utilizan los mismos métodos para controlar el acceso a las réplicas que se utilizan para las tablas de una sola región, incluyendo las políticas de identidad y las políticas basadas en recursos de AWS Identity and Access Management (IAM).

En este tema se describe cómo proteger las tablas globales de DynamoDB mediante permisos de IAM y cifrado de AWS Key Management Service (AWS KMS). Obtendrá información sobre los roles vinculados a servicios (SLR) que permiten la replicación y el escalado automático entre regiones, los permisos de IAM necesarios para crear, actualizar y eliminar tablas globales y las diferencias entre las tablas de coherencia final multirregional (MREC) y multirregional de coherencia sólida (MRSC). También obtendrá información sobre las claves de cifrado de AWS KMS para administrar de forma segura la replicación entre regiones.

Roles vinculados a servicios para tablas globales

Las tablas globales de DynamoDB se basan en roles vinculados a servicios (SLR) para administrar la replicación entre regiones y las capacidades de escalado automático.

Solo tiene que configurar estos roles una vez por cuenta de AWS. Una vez creados, los mismos roles sirven en todas las tablas globales de la cuenta. Para obtener más información acerca los roles vinculados a servicios, consulte Uso de roles vinculados a servicios en la Guía del usuario de IAM.

Rol vinculado a servicios de replicación

Amazon DynamoDB crea automáticamente el rol vinculado al servicio (SLR) `AWSServiceRoleForDynamoDBReplication` cuando se crea la primera tabla global. Este rol administra la replicación entre regiones por usted.

Al aplicar políticas basadas en recursos a las réplicas, asegúrese de no denegar ninguno de los permisos definidos en `AWSServiceRoleForDynamoDBReplicationPolicy` a la entidad principal de SLR, ya que esto interrumpiría la replicación. Si deniega los permisos de SLR necesarios, la replicación hacia y desde las réplicas afectadas se detendrá y el estado de la tabla de réplicas cambiará a `REPLICATION_NOT_AUTHORIZED`.

- Para tablas globales de coherencia final multirregional (MREC), si una réplica permanece en el estado `REPLICATION_NOT_AUTHORIZED` durante más de 20 horas, la réplica se convierte irreversiblemente en una tabla de DynamoDB de una sola región.

- Para las tablas globales de coherencia alta multirregionales (MRSC), la denegación de los permisos necesarios se traduce en `AccessDeniedException` para operaciones de escritura y lectura altamente coherentes. Si una réplica permanece en el estado `REPLICATION_NOT_AUTHORIZED` durante más de siete días, la réplica se vuelve permanentemente inaccesible y las operaciones de escritura y de lectura altamente coherentes seguirán produciendo un error. Algunas operaciones de administración, como la eliminación de réplicas, se realizarán correctamente.

Rol vinculado a un servicio de escalado automático

Al configurar una tabla global para el modo de capacidad aprovisionada, se debe configurar el escalado automático para la tabla global. El escalado automático de DynamoDB utiliza el servicio de escalado automático de aplicaciones de AWS para ajustar dinámicamente la capacidad de rendimiento aprovisionada en las réplicas de tabla global. El servicio de escalado automático de aplicaciones crea un rol vinculado a un servicio (SLR) denominado `AWSServiceRoleForApplicationAutoScaling_DynamoDBTable`. Este rol vinculado al servicio se crea automáticamente en la cuenta de AWS al configurar el escalado automático para una tabla de DynamoDB. Permite al escalado automático de aplicaciones administrar la capacidad de la tabla aprovisionada y crear alarmas de CloudWatch.

Al aplicar políticas basadas en recursos a las réplicas, asegúrese de no denegar ninguno de los permisos definidos en la `AWSApplicationAutoscalingDynamoDBTablePolicy` a la entidad principal de SLR de escalado automático de aplicaciones, ya que esto interrumpiría la funcionalidad de escalado automático.

Políticas de IAM de ejemplo para roles vinculados a servicios

Una política de IAM con la siguiente condición no afecta a los permisos necesarios para la SLR de replicación de DynamoDB y SLR de escalado automático de AWS. Esta condición se puede agregar a políticas que, de otro modo, serían ampliamente restrictivas para evitar interrumpir involuntariamente la replicación o el escalado automático.

Exclusión de los permisos de SLR necesarios de las políticas de denegación

En el siguiente ejemplo se muestra cómo excluir a las entidades principales de roles vinculados al servicio de las instrucciones de denegación:

```
"Condition": {
    "StringNotEquals": {
```

```
        "aws:PrincipalArn": [
            "arn:aws::iam::111122223333:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication",
            "arn:aws::iam::111122223333:role/aws-service-role/dynamodb.application-
autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_DynamoDBTable"
        ]
    }
}
```

Uso de las tablas globales de AWS IAM

En las siguientes secciones se describen los permisos necesarios para las distintas operaciones de las tablas globales y se proporcionan ejemplos de políticas que le ayudarán a configurar el acceso adecuado para los usuarios y aplicaciones.

> **Note**
>
> Todos los permisos descritos se deben aplicar al ARN del recurso de tabla específico en las regiones afectadas. El ARN del recurso de tabla sigue el formato `arn:aws:dynamodb:region:account-id:table/table-name`, en el que debe especificar los valores reales de la región, el ID de cuenta y el nombre de la tabla.

Temas

- Creación de tablas globales y agregación de réplicas
- Actualización de tablas globales
- Eliminación de tablas globales y réplicas

Creación de tablas globales y agregación de réplicas

Las tablas globales de DynamoDB admiten dos modos de coherencia: coherencia final multirregional (MREC) y coherencia alta multirregional (MRSC). Las tablas globales de MREC pueden tener varias réplicas en cualquier número de regiones y proporcionar coherencia final. Las tablas globales de MRSC requieren exactamente tres regiones (tres réplicas o dos réplicas y un testigo) y proporcionan una coherencia sólida con un objetivo de punto de recuperación (RPO).

Los permisos necesarios para crear tablas globales dependen de si se crea una tabla global con o sin un testigo.

Permisos para crear tablas globales

Los siguientes permisos son necesarios para la creación inicial de la tabla global y para agregar réplicas posteriormente. Estos permisos se aplican a las tablas globales de coherencia final multirregional (MREC) y coherencia alta multirregional (MRSC).

- Las tablas globales requieren la replicación entre regiones, que DynamoDB administra mediante el rol vinculado a servicios `AWSServiceRoleForDynamoDBReplication` (SLR). El siguiente permiso permite a DynamoDB crear este rol automáticamente al crear una tabla global por primera vez:
  - `iam:CreateServiceLinkedRole`
- Para crear una tabla global o agregar una réplica mediante la API de `UpdateTable`, debe tener el siguiente permiso en el recurso de la tabla de origen:
  - `dynamodb:UpdateTable`
- Debe tener los siguientes permisos en el recurso de la tabla en las regiones para que se agreguen las réplicas:
  - `dynamodb:CreateTable`
  - `dynamodb:CreateTableReplica`
  - `dynamodb:Query`
  - `dynamodb:Scan`
  - `dynamodb:UpdateItem`
  - `dynamodb:PutItem`
  - `dynamodb:GetItem`
  - `dynamodb:DeleteItem`
  - `dynamodb:BatchWriteItem`

Permisos adicionales para las tablas globales de MRSC mediante un testigo

Al crear una tabla global de coherencia alta multirregional (MRSC) con una región testigo, debe tener el siguiente permiso en el recurso de tabla en todas las regiones participantes (incluidas las regiones réplica y la región testigo):

- `dynamodb:CreateGlobalTableWitness`

Políticas de IAM de ejemplo para crear tablas globales

Creación de una tabla global de MREC o MRSC en tres regiones

La siguiente política basada en la identidad le permite crear una tabla global de MREC o MRSC denominada “usuarios” en tres regiones, incluida la creación del rol vinculado al servicio de replicación de DynamoDB necesario.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreatingUsersGlobalTable",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:CreateTableReplica",
        "dynamodb:UpdateTable",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
        "dynamodb:PutItem",
        "dynamodb:GetItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/users",
        "arn:aws:dynamodb:us-east-2:123456789012:table/users",
        "arn:aws:dynamodb:us-west-2:123456789012:table/users"
      ]
    },
    {
      "Sid": "AllowCreatingSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication"
```

```
      ]
    }
  ]
}
```

Restricción de la creación de tablas globales de MREC o MRSC a regiones específicas

La siguiente política basada en la identidad le permite crear réplicas de tablas globales de DynamoDB en regiones específicas mediante la clave de condición aws:RequestedRegion, incluida la creación del rol vinculado al servicio de replicación de DynamoDB necesario.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAddingReplicasToSourceTable",
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateTable"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": [
            "us-east-1"
          ]
        }
      }
    },
    {
      "Sid": "AllowCreatingReplicas",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:CreateTableReplica",
        "dynamodb:UpdateTable",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
```

```
        "dynamodb:PutItem",
        "dynamodb:GetItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": [
            "us-east-2",
            "us-west-2"
          ]
        }
      }
    },
    {
      "Sid": "AllowCreatingSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication"
      ]
    }
  ]
}
```

Creación de una tabla global de MRSC con un testigo

La siguiente política basada en la identidad le permite crear una tabla global de MRSC de DynamoDB denominada “usuarios” con réplicas en us-east-1 y us-east-2 y un testigo en us-west-2, incluida la creación del rol vinculado al servicio de replicación de DynamoDB necesario.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
```

```
      "Sid": "AllowCreatingUsersGlobalTableWithWitness",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:CreateTableReplica",
        "dynamodb:CreateGlobalTableWitness",
        "dynamodb:UpdateTable",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
        "dynamodb:PutItem",
        "dynamodb:GetItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/users",
        "arn:aws:dynamodb:us-east-2:123456789012:table/users"
      ]
    },
    {
      "Sid": "AllowCreatingSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication"
      ]
    }
  ]
}
```

Restricción de la creación de testigos de MRSC a regiones específicas

Esta política basada en la identidad le permite crear una tabla global de MRSC con réplicas restringidas en regiones específicas mediante la clave de condición aws:RequestedRegion y la creación de testigos sin restricción en todas las regiones, incluida la creación del rol vinculado al servicio de replicación de DynamoDB necesario.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreatingReplicas",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateTable",
        "dynamodb:CreateTableReplica",
        "dynamodb:UpdateTable",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem",
        "dynamodb:PutItem",
        "dynamodb:GetItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": [
            "us-east-1",
            "us-east-2"
          ]
        }
      }
    },
    {
      "Sid": "AllowCreatingWitness",
      "Effect": "Allow",
      "Action": [
        "dynamodb:CreateGlobalTableWitness"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowCreatingSLR",
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
```

```
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/aws-service-role/
replication.dynamodb.amazonaws.com/AWSServiceRoleForDynamoDBReplication"
      ]
    }
  ]
}
```

Actualización de tablas globales

Para modificar la configuración de réplica de una tabla global existente mediante la API de `UpdateTable`, necesita el siguiente permiso en el recurso de tabla de la región en la que va a realizar la llamada a la API:

- `dynamodb:UpdateTable`

Además, puede actualizar otras configuraciones de tablas globales, como las políticas de escalado automático y la configuración del tiempo de vida. Se requieren los siguientes permisos para estas operaciones de actualización adicionales:

- Para actualizar una política de escalado automático de réplicas con la API de `UpdateTableReplicaAutoScaling`, debe tener los siguientes permisos en el recurso de la tabla en todas las regiones que contengan réplicas:
  - `application-autoscaling:DeleteScalingPolicy`
  - `application-autoscaling:DeleteScheduledAction`
  - `application-autoscaling:DeregisterScalableTarget`
  - `application-autoscaling:DescribeScalableTargets`
  - `application-autoscaling:DescribeScalingActivities`
  - `application-autoscaling:DescribeScalingPolicies`
  - `application-autoscaling:DescribeScheduledActions`
  - `application-autoscaling:PutScalingPolicy`
  - `application-autoscaling:PutScheduledAction`
  - `application-autoscaling:RegisterScalableTarget`

- Para actualizar la configuración de Tiempo de vida con la API de `UpdateTimeToLive`, debe tener el siguiente permiso en el recurso de la tabla en todas las regiones que contienen réplicas:
  - `dynamodb:UpdateTimeToLive`

  Tenga en cuenta que el Tiempo de vida (TTL) solo se admite para tablas globales configuradas con coherencia final multirregional (MREC). Para obtener más información sobre cómo funcionan las tablas globales con TTL, consulte Cómo funcionan las tablas globales de DynamoDB.

Eliminación de tablas globales y réplicas

Para eliminar una tabla global, debe eliminar todas las réplicas. Los permisos necesarios para esta operación varían en función de si se elimina una tabla global con o sin una región testigo.

Permisos para eliminar tablas globales y réplicas

Los siguientes permisos son necesarios para eliminar réplicas individuales y para eliminar por completo tablas globales. Al eliminar una configuración de tabla global, solo se elimina la relación de replicación entre tablas de distintas regiones. No se eliminará la tabla de DynamoDB de la última región restante. La tabla de la última región sigue existiendo como tabla estándar de DynamoDB con los mismos datos y configuraciones. Estos permisos se aplican a las tablas globales de coherencia final multirregional (MREC) y coherencia alta multirregional (MRSC).

- Para eliminar réplicas de una tabla global mediante la API de `UpdateTable`, necesita el siguiente permiso en el recurso de tabla de la región desde la que va a realizar la llamada a la API:
  - `dynamodb:UpdateTable`
- Necesita los siguientes permisos en el recurso de tabla en cada región en la que vaya a eliminar una réplica:
  - `dynamodb:DeleteTable`
  - `dynamodb:DeleteTableReplica`

Permisos adicionales para las tablas globales de MRSC mediante un testigo

Para eliminar una tabla global de coherencia alta multirregional (MRSC) con una región testigo, debe tener el siguiente permiso en el recurso de tabla en todas las regiones participantes (incluidas las regiones réplica y la región testigo):

- `dynamodb:DeleteGlobalTableWitness`

Políticas de IAM de ejemplo para eliminar las réplicas de una tabla global

Eliminación de réplicas de tablas globales

Esta política basada en la identidad le permite eliminar una tabla global de DynamoDB denominada “usuarios” y sus réplicas en tres regiones:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateTable",
        "dynamodb:DeleteTable",
        "dynamodb:DeleteTableReplica"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/users",
        "arn:aws:dynamodb:us-east-2:123456789012:table/users",
        "arn:aws:dynamodb:us-west-2:123456789012:table/users"
      ]
    }
  ]
}
```

Eliminación de una tabla global de MRSC con un testigo

Esta política basada en la identidad le permite eliminar la réplica y el testigo de una tabla global de MRSC denominada “usuarios”:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": [
        "dynamodb:UpdateTable",
        "dynamodb:DeleteTable",
        "dynamodb:DeleteTableReplica",
        "dynamodb:DeleteGlobalTableWitness"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/users",
        "arn:aws:dynamodb:us-east-2:123456789012:table/users"
      ]
    }
  ]
}
```

Uso de las tablas globales de AWS KMS

Como todas las tablas de DynamoDB, las réplicas de tablas globales siempre cifran los datos en reposo mediante claves de cifrado almacenadas en AWS Key Management Service (AWS KMS).

Todas las réplicas de una tabla global se deben configurar con el mismo tipo de clave de KMS (clave propia de AWS, clave administrada de AWS o clave administrada por el cliente).

> **Important**
>
> DynamoDB requiere acceso a la clave de cifrado de la réplica para eliminarla. Si desea desactivar o eliminar una clave administrada por el cliente que se utiliza para cifrar una réplica porque va a eliminar la réplica, primero debe eliminar la réplica, esperar a que el estado de la tabla en una de las réplicas restantes cambie a `ACTIVE` y luego desactivar o eliminar la clave.

Para una tabla global configurada para una coherencia final de varias regiones (MREC), si desactiva o revoca el acceso de DynamoDB a una clave administrada por el cliente que se utiliza para cifrar una réplica, la replicación hacia y desde la réplica se detendrá y el estado de la réplica cambiará a `INACCESSIBLE_ENCRYPTION_CREDENTIALS`. Si una réplica de una tabla global de MREC permanece en el estado `INACCESSIBLE_ENCRYPTION_CREDENTIALS` durante más de 20 horas, la réplica se convierte irreversiblemente en una tabla de DynamoDB de una sola región.

Para una tabla global configurada para una coherencia alta de varias regiones (MRSC), si desactiva o revoca el acceso de DynamoDB a una clave administrada por el cliente que se utiliza para cifrar

una réplica, la replicación hacia y desde la réplica se detendrá, los intentos para realizar escrituras o lecturas altamente coherentes en la réplica devolverán un error y el estado de la réplica cambiará a `INACCESSIBLE_ENCRYPTION_CREDENTIALS`. Si una réplica de una tabla global de MRSC permanece en el estado `INACCESSIBLE_ENCRYPTION_CREDENTIALS` durante más de siete días, dependiendo de los permisos específicos revocados, la réplica se archivará o quedará inaccesible permanentemente.

Descripción de la facturación de tablas globales en Amazon DynamoDB

En esta guía se describe cómo funciona la facturación de DynamoDB para las tablas globales, se identifican los componentes que contribuyen al costo de las tablas globales y se incluye un ejemplo práctico.

Las tablas globales de Amazon DynamoDB son una base de datos totalmente administrada, sin servidor, multirregional y multiactiva. Las tablas globales están diseñadas para ofrecer una disponibilidad del 99,999 %, una mayor resiliencia de las aplicaciones y una mejor continuidad empresarial. Las tablas globales replican automáticamente las tablas de DynamoDB en las regiones de AWS que elija para que pueda lograr un rendimiento de lectura y escritura rápido a escala local.

Funcionamiento

El modelo de facturación de las tablas globales difiere del de las tablas de DynamoDB de una sola región. Las operaciones de escritura de las tablas de DynamoDB de una sola región se facturan mediante las siguientes unidades:

- Unidades de solicitud de escritura (WRU) para el modo de capacidad bajo demanda, en el que se cobra una WRU por cada escritura de hasta 1 KB
- Unidades de capacidad de escritura (WCU) para el modo de capacidad aprovisionada, en el que una WCU proporciona una escritura por segundo de hasta 1 KB

Al crear una tabla global mediante la adición de una tabla de réplica a una tabla de una sola región existente, dicha tabla se convierte en una tabla de réplica, lo que significa que también cambian las unidades que se usan para facturar las escrituras en la tabla. Las operaciones de escritura en las tablas de réplica se facturan mediante las siguientes unidades:

- Unidades de solicitud de escritura replicada (rWRU) para el modo de capacidad bajo demanda, en el que se cobra una rWRU por tabla de réplica por cada escritura de hasta 1 KB

- Unidades de capacidad de escritura replicada (rWCU) para el modo de capacidad aprovisionada, en el que una WCU por tabla de réplica proporciona una escritura por segundo de hasta 1 KB

Las actualizaciones de los índices secundarios globales (GSI) se facturan mediante las mismas unidades que las tablas de DynamoDB de una sola región, aunque la tabla base del GSI sea una tabla de réplica. Las operaciones de actualización de los GSI se facturan mediante las siguientes unidades:

- Unidades de solicitud de escritura (WRU) para el modo de capacidad bajo demanda, en el que se cobra una WRU por cada escritura de hasta 1 KB
- Unidades de capacidad de escritura (WCU) para el modo de capacidad aprovisionada, en el que una WCU proporciona una escritura por segundo de hasta 1 KB

Las unidades de escritura replicada (rWCU y rWRU) tienen el mismo precio que las unidades de escritura de una sola región (WCU y WRU). Se aplican tarifas de transferencia de datos entre regiones a las tablas globales, ya que los datos se replican entre regiones. Los cargos por escritura replicada (rWCU o rWRU) se producen en cada región que contenga una tabla de réplica para la tabla global.

Las operaciones de lectura de tablas de una sola región y de tablas de réplica utilizan las siguientes unidades:

- Unidades de solicitud de lectura (RRU) para el modo de capacidad bajo demanda, en el que se cobra una RRU por cada lectura altamente coherente de hasta 4 KB
- Unidades de capacidad de lectura (RCU) para tablas aprovisionadas, en las que una RCU proporciona una lectura altamente coherente por segundo de hasta 4 KB

Modos de coherencia y facturación

Las unidades de escritura replicadas (rWCU y rWRU) que se utilizan para facturar las operaciones de escritura son idénticas en los modos de coherencia sólida de varias regiones (MRSC) y de coherencia final de varias regiones (MREC). Las tablas globales que utilizan el modo de coherencia alta multirregional (MRSC) configurado con un testigo no incurren en costos de unidades de escritura replicadas (rWCU y rWRU), costos de almacenamiento o costos de transferencia de datos para la replicación al testigo.

Ejemplo de facturación de tablas globales de DynamoDB

Veamos un ejemplo de varios días para comprobar cómo funciona en la práctica la facturación de las solicitudes de escritura de tablas globales (tenga en cuenta que en este ejemplo solo se consideran las solicitudes de escritura y no se incluyen los gastos de restauración de tablas ni de transferencia de datos entre regiones en los que se incurriría en el ejemplo):

Día 1, tabla de una sola región: tiene una tabla de DynamoDB bajo demanda de una sola región llamada Table_A en la región us-west-2. Escribe 100 elementos de 1 KB en Table_A. Para estas operaciones de escritura de una sola región, se le cobrará 1 unidad de solicitud de escritura (WRU) por cada 1 KB escrito. Los cargos del día 1 son los siguientes:

- 100 WRU en la región us-west-2 por escrituras de una sola región

El total de unidades de solicitud cobradas el día 1 es de 100 WRU.

Día 2, creación de una tabla global: para crear una tabla global, agrega una réplica de Table_A a la región us-east-2. Table_A es ahora una tabla global con dos tablas de réplica: una en la región us-west-2 y otra en la región us-east-2. Escribe 150 elementos de 1 KB en la tabla de réplica de la región us-west-2. Los cargos del día 2 son los siguientes:

- 150 rWRU en la región us-west-2 para escrituras replicadas
- 150 rWRU en la región us-east-2 para escrituras replicadas

El total de unidades de solicitud cobradas el día 2 es de 300 rWRU.

Día 3, adición de un índice secundario global: agrega un índice secundario global (GSI) a la tabla de réplica de la región us-east-2 que proyecta todos los atributos de la tabla base (réplica). La tabla global crea automáticamente para usted el GSI en la tabla de réplica de la región us-west-2. Escribe 200 registros nuevos de 1 KB en la tabla de réplica de la región us-west-2. Los cargos del día 3 son los siguientes:

- • 200 rWRU en la región us-west-2 para escrituras replicadas
- • 200 WRU en la región us-west-2 para las actualizaciones de GSI
- • 200 rWRU en la región us-east-2 para escrituras replicadas
- • 200 WRU en la región us-east-2 para las actualizaciones de GSI

El total de unidades de solicitud de escritura cobradas el día 3 es 400 WRU y 400 rWRU.

Los cargos totales por unidad de escritura durante los tres días son de 500 rWRU (100 rWRU el día 1 + 400 rWRU el día 3) y 700 rWRU (300 rWRU el día 2 + 400 rWRU el día 3).

En resumen, las operaciones de escritura de tabla de réplica se facturan en unidades de escritura replicada en todas las regiones que contienen una tabla de réplica. Si tiene índices secundarios globales, se le cobran unidades de escritura por las actualizaciones de los GSI en todas las regiones que contienen un GSI (que en una tabla global son todas las regiones que contienen una tabla de réplica).

Versiones de tablas globales de DynamoDB

Hay disponibles dos versiones de las tablas globales de DynamoDB: versión 2019.11.21 (actual) y versión 2017.11.29 (antigua). Recomendamos utilizar la versión 2019.11.21 (actual) de las tablas globales, ya que es más fácil de usar, se admite en más regiones y tiene un costo más bajo para la mayoría de las cargas de trabajo en comparación con la versión 2017.11.29 (antigua).

Determinación de la versión de una tabla global

Determinar la versión mediante la AWS CLI

Identificación de una réplica de tabla global de la versión 2019.11.21 (actual)

Para determinar si una tabla es una réplica de tabla global de la versión 2019.11.21 (actual), invoque el comando `describe-table` para la tabla. Si el resultado contiene el atributo de `GlobalTableVersion` con un valor de “2019.11.21”, la tabla es una réplica de tabla global de la versión 2019.11.21 (actual).

Un ejemplo de comando de la CLI para `describe-table`:

```
aws dynamodb describe-table \
--table-name users \
--region us-east-2
```

El resultado (abreviado) contiene el atributo de `GlobalTableVersion` con un valor de “2019.11.21”, por lo que esta tabla es una réplica de la tabla global de la versión 2019.11.21 (actual).

```
{
    "Table": {
```

```
        "AttributeDefinitions": [
            {
                "AttributeName": "id",
                "AttributeType": "S"
            },
            {
                "AttributeName": "name",
                "AttributeType": "S"
            }
        ],
        "TableName": "users",
        ...
        "GlobalTableVersion": "2019.11.21",
        "Replicas": [
            {
                "RegionName": "us-west-2",
                "ReplicaStatus": "ACTIVE",
            }
        ],
        ...
    }
}
```

Identificación de una réplica de tabla global de la versión 2017.11.29 (antigua)

La versión 2017.11.29 (antigua) de tablas globales utiliza un conjunto dedicado de comandos para la administración de tablas globales. Para determinar si una tabla es una réplica de la versión 2017.11.29 de tablas globales (antigua), invoque el comando `describe-global-table` para la tabla. Si recibe una respuesta exitosa, la tabla es una réplica de tabla global de la versión 2017.11.29 (antigua). Si el comando `describe-global-table` devuelve un error `GlobalTableNotFoundException`, la tabla no es una réplica de la versión 2017.11.29 (antigua).

Un comando de la CLI de ejemplo para `describe-global-table`:

```
aws dynamodb describe-global-table \
--table-name users \
--region us-east-2
```

El comando devuelve una respuesta exitosa, por lo que esta tabla es una réplica de tabla global de la versión 2017.11.29 (antigua).

```
{
```

```
    "GlobalTableDescription": {
        "ReplicationGroup": [
            {
                "RegionName": "us-west-2"
            },
            {
                "RegionName": "us-east-2"
            }
        ],
        "GlobalTableArn": "arn:aws:dynamodb::123456789012:global-table/users",
        "CreationDateTime": "2025-06-10T13:55:53.630000-04:00",
        "GlobalTableStatus": "ACTIVE",
        "GlobalTableName": "users"
    }
}
```

Determinación de la versión mediante la consola de DynamoDB

Para identificar la versión de una réplica de tabla global, realice lo siguiente:

1. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/home.
2. En el panel de navegación del lado izquierdo de la consola, elija Tablas.
3. Elija la tabla de la que desea identificar la versión de las tablas globales.
4. Elija la pestaña Global Tables (Tablas globales).

   La sección Resumen muestra la versión de las tablas globales en uso.

Diferencias de comportamiento entre las versiones heredada y actual

En la siguiente lista se describen las diferencias de comportamiento entre las versiones heredada y actual de las tablas globales.

- La versión 2019.11.21 (actual) consume menos capacidad de escritura para varias operaciones de DynamoDB en comparación con la 2017.11.29 (heredada) y, por lo tanto, es más rentable para la mayoría de los clientes. Las diferencias entre estas operaciones de DynamoDB son las siguientes:
  - La invocación de PutItem para un elemento de 1 KB en una región y replicarlo en otras regiones requiere dos rWRU por región para la versión 2017.11.29 (heredada), pero solo un rWrU para la 2019.11.21 (actual).

- La invocación de UpdateItem para un elemento de 1 KB requiere dos rWRU en la región de origen y un rWrU por región de destino para la versión 2017.11.29 (heredada), pero solo un rWrU para las regiones de origen y destino para la 2019.11.21 (actual).
- La invocación de DeleteItem para un elemento de 1 KB requiere un rWRU en la región de origen y dos rWrU por región de destino para la versión 2017.11.29 (heredada), pero solo un rWrU para las regiones de origen o destino para la 2019.11.21 (actual).

La siguiente tabla muestra el consumo de rWRU de las tablas de la versión 2017.11.29 (heredada) y de la 2019.11.21 (actual) para un elemento de 1 KB en dos regiones.

Operación	2017.11.29 (heredada)	2019.11.21 (actual)	Ahorro
PutItem	4 rWRU	2 rWRU	50%
UpdateItem	3 rWRU	2 rWRU	33 %
DeleteItem	3 rWRU	2 rWRU	33 %

- La versión 2017.11.29 (heredada) solo está disponible en 11 Regiones de AWS. Sin embargo, la versión 2019.11.21 (actual) está disponible en todas las Regiones de AWS.
- Para crear tablas globales de la versión 2017.11.29 (heredada), primero debe crear un conjunto de tablas regionales vacías y, a continuación, invocar la API CreateGlobalTable para formar la tabla global. Para crear tablas globales de la versión 2019.11.21 (actual), invoque la API UpdateTable para agregar una réplica a una tabla regional existente.
- La versión 2017.11.29 (heredada) requiere que se vacíen todas las réplicas de la tabla antes de agregar una réplica en una nueva región (incluso durante la creación). La versión 2019.11.21 (actual) permite agregar y eliminar réplicas de regiones de una tabla que ya contenga datos.
- La versión 2017.11.29 (heredada) utiliza el siguiente conjunto dedicado de API de plano de control para administrar las réplicas:
  - CreateGlobalTable
  - DescribeGlobalTable
  - DescribeGlobalTableSettings
  - ListGlobalTables
  - UpdateGlobalTable
  - UpdateGlobalTableSettings

La versión 2019.11.21 (actual) usa las API DescribeTable y UpdateTable para administrar las réplicas.

- La versión 2017.11.29 (heredada) publica dos registros de DynamoDB Streams para cada escritura. La versión 2019.11.21 (actual) solo publica un registro de DynamoDB Streams para cada escritura.
- La versión 2017.11.29 (heredada) rellena y actualiza los atributos `aws:rep:deleting`, `aws:rep:updateregion` y `aws:rep:updatetime`. La versión 2019.11.21 (actual) no rellena ni actualiza estos atributos.
- La versión 2017.11.29 (heredada) no sincroniza la configuración Uso del período de vida (TTL) en DynamoDB entre las réplicas. La versión 2019.11.21 (actual) sincroniza la configuración de TTL entre las réplicas.
- La versión 2017.11.29 (heredada) no replica las eliminaciones de TTL en otras réplicas. La versión 2019.11.21 (actual) replica las eliminaciones de TTL en todas las réplicas.
- La versión 2017.11.29 (heredada) no sincroniza la configuración del escalado automático entre las réplicas. La versión 2019.11.21 (actual) sincroniza la configuración del escalado automático entre las réplicas.
- La versión 2017.11.29 (heredada) no sincroniza la configuración del índice secundario global (GSI) entre las réplicas. La versión 2019.11.21 (actual) sincroniza la configuración de GSI entre las réplicas.
- La versión 2017.11.29 (heredada) no sincroniza la configuración del cifrado en reposo entre las réplicas. La versión 2019.11.21 (actual) sincroniza la configuración del cifrado en reposo entre las réplicas.
- La versión 2017.11.29 (heredada) publica la métrica `PendingReplicationCount`. La versión 2019.11.21 (actual) no publica esta métrica.

Actualización a la versión actual

Permisos necesarios para actualizar las tablas globales

Para actualizar a la versión 2019.11.21 (actual), debe tener permisos `dynamodb:UpdateGlobalTableversion` en todas las regiones que tengan réplicas. Estos permisos se suman a los permisos necesarios para acceder a la consola de DynamoDB y ver las tablas.

La siguiente política de IAM concede permisos para actualizar cualquier tabla global a la versión 2019.11.21 (actual).

```
{
    "version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:UpdateGlobalTableversion",
            "Resource": "*"
        }
    ]
}
```

La siguiente política de IAM concede permisos para actualizar solo la tabla global `Music`, con réplicas en dos regiones, a la versión 2019.11.21 (actual).

```
{
    "version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:UpdateGlobalTableversion",
            "Resource": [
                "arn:aws:dynamodb::123456789012:global-table/Music",
                "arn:aws:dynamodb:ap-southeast-1:123456789012:table/Music",
                "arn:aws:dynamodb:us-east-2:123456789012:table/Music"
            ]
        }
    ]
}
```

Qué esperar durante la actualización

- Todas las réplicas de tablas globales seguirán procesando el tráfico de lectura y escritura durante la actualización.
- El proceso de actualización tarda entre unos minutos a varias horas, según el tamaño de la tabla y la cantidad de réplicas.
- Durante el proceso de actualización, el valor de TableStatus cambiará de `ACTIVE` a `UPDATING`. Puede ver el estado de la tabla invocando la API DescribeTable o con la vista Tablas de la consola de DynamoDB.

- El escalado automático no ajustará la configuración de capacidad aprovisionada para una tabla global mientras se esté actualizando la tabla. Le recomendamos encarecidamente que configure la tabla en el modo de capacidad bajo demanda durante la actualización.
- Si decide utilizar capacidad aprovisionada con el escalado automático durante la actualización, debe aumentar el rendimiento mínimo de lectura y escritura en las políticas para contemplar el aumento esperado de tráfico y evitar cualquier limitación durante la actualización.
- La métrica `ReplicationLatency` puede informar temporalmente de picos de latencia o dejar de informar de datos métricos durante el proceso de actualización. Para obtener más información, consulte the section called “ReplicationLatency”.
- Cuando finaliza el proceso de actualización, el estado de la tabla cambia a `ACTIVE`.

Comportamiento de DynamoDB Streams antes, durante y después de la actualización

Operación	Regiones de réplica	Comportamiento antes de la actualización	Comportamiento durante la actualización	Comportamiento tras la actualización
Put o Update	Origen	Las marcas de tiempo se rellenan mediante UpdateItem.	Las marcas de tiempo se rellenan mediante PutItem.	No se genera ninguna marca de tiempo visible para el cliente.
		Se generan dos registros de Streams. El primer registro contiene los atributos escritos por el cliente. El segundo registro contiene los atributos `aws:rep:*` .	Se generan dos registros de Streams. El primer registro contiene los atributos escritos por el cliente. El segundo registro contiene los atributos `aws:rep:*` .	Se genera un único registro de Streams que contiene los atributos escritos por el cliente.

Operación	Regiones de réplica	Comportamiento antes de la actualización	Comportamiento durante la actualización	Comportamiento tras la actualización
		Se consumen dos rWCU por cada escritura del cliente.	Se consumen dos rWCU por cada escritura del cliente.	Se consume una rWCU por cada escritura del cliente.
		Las métricas `ReplicationLatency` y `PendingReplicationCount` se publican en CloudWatch.	Las métricas `ReplicationLatency` y `PendingReplicationCount` se publican en CloudWatch.	La métrica `ReplicationLatency` se publica en CloudWatch.
	Destino	La replicación se realiza mediante PutItem.	La replicación se realiza mediante PutItem.	La replicación se realiza mediante PutItem.
		Se genera un único registro de Streams, que contiene tanto los atributos escritos por el cliente como los atributos `aws:rep:*` .	Se genera un único registro de Streams, que contiene tanto los atributos escritos por el cliente como los atributos `aws:rep:*` .	Se genera un único registro de Streams, que solo contiene los atributos escritos por el cliente sin atributos de replicación.

Operación	Regiones de réplica	Comportamiento antes de la actualización	Comportamiento durante la actualización	Comportamiento tras la actualización
		Se consume una rWCU si el elemento existe en la región de destino. Se consumen dos rWCU si el elemento no existe en la región de destino.	Se consume una rWCU si el elemento existe en la región de destino. Se consumen dos rWCU si el elemento no existe en la región de destino.	Se consume una rWCU por cada escritura del cliente.
		Las métricas `ReplicationLatency` y `PendingReplicationCount` se publican en CloudWatch.	Las métricas `ReplicationLatency` y `PendingReplicationCount` se publican en CloudWatch.	La métrica `ReplicationLatency` se publica en CloudWatch.
Delete	Origen	Elimine cualquier elemento con una marca de tiempo más pequeña con DeleteItem.	Elimine cualquier elemento con una marca de tiempo más pequeña con DeleteItem.	Elimine cualquier elemento con una marca de tiempo más pequeña con DeleteItem.

Operación	Regiones de réplica	Comportamiento antes de la actualización	Comportamiento durante la actualización	Comportamiento tras la actualización
		Se genera un único registro de Streams, que contiene tanto los atributos escritos por el cliente como los atributos `aws:rep:*` .	Se genera un único registro de Streams, que contiene tanto los atributos escritos por el cliente como los atributos `aws:rep:*` .	Se genera un único registro de Streams, que contiene los atributos escritos por el cliente.
		Se consume una rWCU por cada eliminación del cliente.	Se consume una rWCU por cada eliminación del cliente.	Se consume una rWCU por cada eliminación del cliente.
		Las métricas `ReplicationLatency` y `PendingReplicationCount` se publican en CloudWatch.	Las métricas `ReplicationLatency` y `PendingReplicationCount` se publican en CloudWatch.	La métrica `ReplicationLatency` se publica en CloudWatch.

Operación	Regiones de réplica	Comportamiento antes de la actualización	Comportamiento durante la actualización	Comportamiento tras la actualización
	Destino	Se realizan eliminaciones en dos fases: • En la fase 1, UpdateItem establece la marca de borrado. • En la fase 2, DeleteItem elimina el elemento.	Elimina el elemento mediante DeleteItem.	Elimina el elemento mediante DeleteItem.
		Se generan dos registros de Streams. El primer registro contiene el cambio en el campo `aws:rep:deleting` . El segundo registro contiene los atributos escritos por el cliente y los atributos.	Se genera un único registro de Streams, que contiene los atributos escritos por el cliente.	Se genera un único registro de Streams, que contiene los atributos escritos por el cliente.
		Se consumen dos rWCU por cada eliminación del cliente.	Se consume una rWCU por cada eliminación del cliente.	Se consume una rWCU por cada eliminación del cliente.

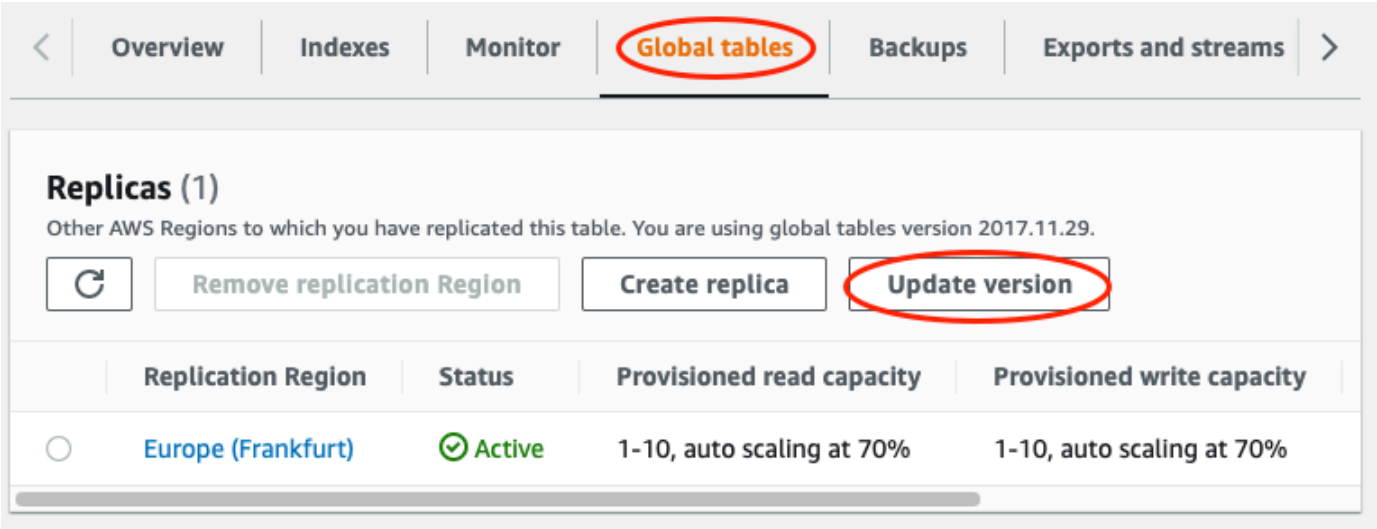
Operación	Regiones de réplica	Comportamiento antes de la actualización	Comportamiento durante la actualización	Comportamiento tras la actualización
		Las métricas `ReplicationLatency` y `PendingReplicationCount` se publican en CloudWatch.	La métrica `ReplicationLatency` se publica en CloudWatch.	La métrica `ReplicationLatency` se publica en CloudWatch.

Actualización a la versión 2019.11.21 (actual)

Siga estos pasos para actualizar la versión de las tablas globales de DynamoDB mediante la Consola de administración de AWS.

Actualización de las tablas globales a la versión 2019.11.21 (actual)

1. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/home.
2. En el panel de navegación del lado izquierdo de la consola, elija Tablas y, a continuación, seleccione la tabla global que desea actualizar a la versión 2019.11.21 (actual).
3. Elija la pestaña Global Tables (Tablas globales).
4. Elija Update version (Actualizar versión).



5. Lea y acepte los nuevos requisitos y, a continuación, elija Update version (Actualizar versión).
6. Una vez completado el proceso de actualización, la versión de las tablas globales que aparece en la consola cambia a 2019.11.21.

Prácticas recomendadas para tablas globales

En las siguientes secciones, se describen las prácticas recomendadas para implementar y utilizar las tablas globales.

Versión

Hay disponibles dos versiones de las tablas globales de DynamoDB: versión 2019.11.21 (actual) y versión 2017.11.29 (antigua). Debe utilizar la versión 2019.11.21 (actual) siempre que sea posible.

Protección contra eliminación

Debe habilitar la protección contra eliminación en las réplicas de tablas globales que desee proteger contra la eliminación accidental. Debe habilitar la protección contra la eliminación en cada réplica.

Uso de AWS CloudFormation

CloudFormation actualmente no admite la coordinación de recursos de varias regiones, como tablas globales entre pilas. Si define cada réplica de una tabla global en una pila regional independiente, se producirán errores debido a la desviación detectada entre las pilas al realizar actualizaciones de réplicas. Para evitar este problema, debe elegir una región como región de referencia para implementar las tablas globales y definir todas las réplicas de la tabla global en la pila de esa región.

> **Important**
>
> No puede convertir un recurso de tipo `AWS::DynamoDB::Table` en un recurso de tipo `AWS::DynamoDB::GlobalTable` cambiando su tipo en la plantilla. Intentar convertir una tabla de una sola región en una tabla global cambiando el tipo de recurso de CloudFormation puede resultar en la eliminación de la tabla de DynamoDB.

Puede utilizar el recurso de `AWS::DynamoDB::GlobalTable` para crear una tabla en una sola región. Esta tabla se implementará como cualquier otra tabla de una sola región. Si posteriormente actualiza la pila para agregar otras regiones a un recurso, las réplicas se agregarán a la tabla y esta se convertirá de forma segura en una tabla global.

Si tiene un recurso de `AWS::DynamoDB::Table` existente que desea convertir en un recurso de `AWS::DynamoDB::GlobalTable`, los pasos recomendados para convertir el tipo de recurso son los siguientes:

1. Establezca la política de eliminación de `AWS::DynamoDB::Table` para retenerla.
2. Elimine la tabla de la definición de la pila.
3. Agregue réplicas a la tabla de una sola región en la consola de AWS y conviértala en una tabla global.
4. Importe la nueva tabla global como un nuevo recurso de `AWS::DynamoDB::GlobalTable` a la pila.

Copias de seguridad y recuperación en un momento dado

La habilitación de las copias de seguridad automatizadas y la recuperación en un momento dado (PITR) para una réplica en una tabla global puede ser suficiente para cumplir los objetivos de recuperación ante desastres. Las copias de seguridad de réplicas creadas con copia de seguridad de AWS se pueden replicar automáticamente en todas las regiones para una mayor resiliencia. Tenga en cuenta los objetivos del plan de recuperación ante desastres en el contexto de la alta disponibilidad en varias regiones al elegir su estrategia de copia de seguridad y habilitación de PITR.

Diseño para alta disponibilidad de varias regiones

Para obtener instrucciones prescriptivas sobre la implementación de tablas globales, consulte Prácticas recomendadas para el diseño de tablas globales de DynamoDB.

Uso de elementos y atributos en DynamoDB

En Amazon DynamoDB, un elemento es una colección de atributos. Cada atributo tiene un nombre y un valor. Los valores de los atributos pueden ser escalares, conjuntos o tipos de documentos. Para obtener más información, consulte Funcionamiento de Amazon DynamoDB.

DynamoDB proporciona cuatro operaciones que aportan la funcionalidad básica de creación, lectura, actualización y eliminación (CRUD). Todas estas operaciones son atómicas.

- `PutItem`: permite crear un elemento.
- `GetItem`: permite leer un elemento.
- `UpdateItem`: permite actualizar un elemento.

- `DeleteItem`: permite eliminar un elemento.

Cada una de estas operaciones requiere que especifique la clave principal del elemento que se va a usar. Por ejemplo, para leer un elemento mediante `GetItem`, debe especificar la clave de partición y la clave de ordenación (si procede) de ese elemento.

Además de las cuatro operaciones CRUD básicas, DynamoDB también ofrece las siguientes:

- `BatchGetItem`: permite leer hasta 100 elementos de una o varias tablas.
- `BatchWriteItem`: permite crear o eliminar hasta 25 elementos en una o varias tablas.

Estas operaciones por lotes combinan varias operaciones CRUD en una sola solicitud. Además, las operaciones por lotes leen y escriben los elementos en paralelo, para minimizar las latencias de respuesta.

Esta sección se describe cómo utilizar estas operaciones y se incluyen los temas relacionados, tales como las actualizaciones condicionales y los contadores atómicas. Además, se facilitan ejemplos de código en los que se utilizan los SDK de AWS.

Temas

- Tamaños y formatos de elementos de DynamoDB
- Lectura de un elemento
- Escritura de un elemento
- Valores devueltos
- Operaciones por lotes
- Contadores atómicos
- Escrituras condicionales
- Uso de expresiones en DynamoDB
- Uso del período de vida (TTL) en DynamoDB
- Consulta de tablas en DynamoDB
- Análisis de tablas en DynamoDB
- PartiQL: un lenguaje de consulta compatible con SQL para Amazon DynamoDB
- Uso de elementos: Java
- Uso de elementos: .NET

Tamaños y formatos de elementos de DynamoDB

Las tablas de DynamoDB no tienen esquemas, salvo la clave principal. Por este motivo, todos los elementos de una tabla pueden tener atributos, tamaños y tipos de datos distintos.

El tamaño total de un elemento es la suma de las longitudes de los nombres y los valores de sus atributos, además de cualquier gasto adicional aplicable, como se describe a continuación. Puede utilizar las siguientes directrices para calcular el tamaño de los atributos:

- Los valores de tipo String son Unicode con codificación binaria UTF-8. El tamaño de una cadena es (número de bytes con codificación UTF-8 del nombre de atributo) + (número de bytes con codificación UTF-8).
- Los números son de longitud variable, con un máximo de 38 dígitos significativos. Los ceros iniciales y finales se recortan. El tamaño aproximado de un número es (número de bytes con codificación UTF-8 del nombre de atributo) + (1 byte por cada dos dígitos significativos) + (1 byte).
- Un valor binario se debe codificar previamente en formato base64 para poder enviarlo a DynamoDB. Sin embargo, para calcular su tamaño se utiliza la longitud en bytes sin procesar del valor. El tamaño de un atributo binario es (número de bytes con codificación UTF-8 del nombre de atributo) + (número de bytes sin procesar).
- El tamaño de un atributo nulo o booleano es (número de bytes con codificación UTF-8 del nombre de atributo) + (1 byte).
- Un atributo de tipo `List` o `Map` requiere 3 bytes adicionales, independientemente de su contenido. El tamaño de un atributo `List` o `Map` es (número de bytes con codificación UTF-8 del nombre de atributo) + suma (tamaño de los elementos anidados) + (3 bytes). El tamaño de un atributo `List` o `Map` vacío es (número de bytes con codificación UTF-8 del nombre de atributo) + (3 bytes).
- Cada elemento `List` o `Map` también requiere 1 byte de sobrecarga.

> **Note**
>
> Recomendamos elegir nombres de atributos cortos en lugar de largos. Esto le ayuda a reducir la cantidad de almacenamiento necesario, así como la cantidad de RCU/WCU que utiliza.

A efectos de facturación del almacenamiento, cada elemento incluye una capacidad de almacenamiento adicional por elemento que depende de las características que haya habilitado.

- Todos los elementos de DynamoDB requieren 100 bytes de capacidad de almacenamiento adicional para la indexación.
- Algunas características de DynamoDB (tablas globales, transacciones, captura de datos de cambios para Kinesis Data Streams con DynamoDB) requieren una capacidad de almacenamiento adicional para tener en cuenta los atributos creados por el sistema como resultado de la habilitación de dichas características. Por ejemplo, las tablas globales requieren 48 bytes adicionales de capacidad de almacenamiento.

Lectura de un elemento

Para leer un elemento de una tabla de DynamoDB, se utiliza la operación `GetItem`. Debe proporcionar el nombre de la tabla, así como la clave principal del elemento que se desea.

Example

En el siguiente ejemplo de la AWS CLI se muestra cómo leer un elemento de la tabla `ProductCatalog`.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}'
```

> **Note**
>
> Con `GetItem`, es preciso especificar la clave principal completa, no solo una parte de ella. Por ejemplo, si una tabla contiene una clave principal compuesta (clave de partición y clave de ordenación), tendrá que proporcionar un valor para la clave de partición y un valor para la clave de ordenación.

De forma predeterminada, una solicitud `GetItem` lleva a cabo una lectura consistente final. Puede usar el parámetro `ConsistentRead` para solicitar una lectura de consistencia alta, si lo prefiere. (Esto consume unidades de capacidad de lectura adicionales, pero devuelve la versión más actualizada del elemento).

`GetItem` devuelve todos los atributos del elemento. Puede usar una expresión de proyección para devolver solamente algunos de los atributos. Para obtener más información, consulte Uso de expresiones de proyección en DynamoDB.

Para devolver el número de unidades de capacidad de lectura consumidas por `GetItem`, establezca el parámetro `ReturnConsumedCapacity` en `TOTAL`.

Example

En el siguiente ejemplo de la AWS Command Line Interface (AWS CLI) se muestran algunos de los parámetros de `GetItem` opcionales.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --consistent-read \
    --projection-expression "Description, Price, RelatedItems" \
    --return-consumed-capacity TOTAL
```

Escritura de un elemento

Para crear, actualizar o eliminar un elemento de una tabla de DynamoDB, utilice una de las siguientes operaciones:

- `PutItem`
- `UpdateItem`
- `DeleteItem`

Para cada una de estas operaciones, debe especificar la clave principal completa, no solo parte de ella. Por ejemplo, si una tabla contiene una clave principal compuesta (clave de partición y clave de ordenación), tendrá que proporcionar un valor para la clave de partición y un valor para la clave de ordenación.

Para devolver el número de unidades de capacidad de escritura consumidas por cualquiera de estas operaciones, establezca el parámetro `ReturnConsumedCapacity` en uno de los valores siguientes:

- `TOTAL`: devuelve el número total de unidades de capacidad de escritura consumidas.
- `INDEXES`: devuelve el número total de unidades de capacidad de escritura consumidas, con subtotales para la tabla y todos los índices secundarios que se hayan visto afectados por la operación.
- `NONE`: no devuelve ningún dato de capacidad de escritura consumida. (Esta es la opción predeterminada.)

PutItem

`PutItem` crea un elemento nuevo. Si ya existe un elemento con la misma clave en la tabla, se sustituirá por el nuevo.

Example

Escriba un nuevo elemento en la tabla `Thread`. La clave principal de `Thread` consta de `ForumName` (clave de partición) y `Subject` (clave de ordenación).

```
aws dynamodb put-item \
    --table-name Thread \
    --item file://item.json
```

Los argumentos de `--item` se almacenan en el archivo `item.json`.

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"},
    "Message": {"S": "First post in this thread"},
    "LastPostedBy": {"S": "fred@example.com"},
    "LastPostDateTime": {"S": "201603190422"}
}
```

UpdateItem

Si no existe un elemento con la clave especificada, `UpdateItem` crea uno nuevo. De lo contrario, modifica los atributos de un elemento existente.

Se utiliza una expresión de actualización para especificar los atributos que se desea modificar y los nuevos valores. Para obtener más información, consulte Uso de expresiones de actualización en DynamoDB.

En la expresión de actualización, se utilizan valores de atributos de expresión como marcadores de posición de los valores reales. Para obtener más información, consulte Uso de valores de atributos de expresión en DynamoDB.

Example

Modifique varios atributos del elemento `Thread`. El parámetro `ReturnValues` opcional muestra el elemento tal y como aparece después de la actualización. Para obtener más información, consulte Valores devueltos.

```
aws dynamodb update-item \
    --table-name Thread \
    --key file://key.json \
    --update-expression "SET Answered = :zero, Replies = :zero, LastPostedBy
 = :lastpostedby" \
    --expression-attribute-values file://expression-attribute-values.json \
    --return-values ALL_NEW
```

Los argumentos de `--key` se almacenan en el archivo `key.json`.

```
{
    "ForumName": {"S": "Amazon DynamoDB"},
    "Subject": {"S": "New discussion thread"}
}
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo `expression-attribute-values.json`.

```
{
    ":zero": {"N":"0"},
    ":lastpostedby": {"S":"barney@example.com"}
}
```

DeleteItem

`DeleteItem` elimina el elemento con la clave especificada.

Example

En el ejemplo siguiente de la AWS CLI, se muestra cómo eliminar el elemento `Thread`.

```
aws dynamodb delete-item \
    --table-name Thread \
    --key file://key.json
```

Valores devueltos

En algunos casos, es posible que desee que DynamoDB devuelva los valores de algunos atributos tal y como aparecen antes o después de modificarlos. Las operaciones `PutItem`, `UpdateItem` y

`DeleteItem` tienen un parámetro `ReturnValues` que se puede usar para devolver los valores de los atributos antes o después de modificarlos.

El valor predeterminado de `ReturnValues` es `NONE`, es decir, DynamoDB no devuelve ninguna información sobre los atributos modificados.

A continuación se indican la otra configuración válida de `ReturnValues`, organizados según la operación de la API de DynamoDB.

PutItem

- `ReturnValues`: `ALL_OLD`
  - Si sobrescribe un elemento existente, `ALL_OLD` devuelve el elemento completo tal y como aparecía antes de sobrescribirlo.
  - Si escribe un elemento que no existía, `ALL_OLD` no surte efecto.

UpdateItem

Lo más frecuente es usar `UpdateItem` para actualizar un elemento existente. Sin embargo, `UpdateItem` en realidad lleva a cabo una operación upsert (actualización/inserción). Esto quiere decir que creará el elemento automáticamente si este no existe.

- `ReturnValues`: `ALL_OLD`
  - Si actualiza un elemento existente, `ALL_OLD` devuelve el elemento completo tal y como aparecía antes de actualizarlo.
  - Si actualiza un elemento que no existía (upsert), `ALL_OLD` no surte efecto.
- `ReturnValues`: `ALL_NEW`
  - Si actualiza un elemento existente, `ALL_NEW` devuelve el elemento completo tal y como aparece después de actualizarlo.
  - Si actualiza un elemento que no existía (upsert), `ALL_NEW` devuelve el elemento completo.
- `ReturnValues`: `UPDATED_OLD`
  - Si actualiza un elemento existente, `UPDATED_OLD` devuelve solamente los atributos actualizados, tal y como aparecían antes de la actualización.
  - Si actualiza un elemento que no existía (upsert), `UPDATED_OLD` no surte efecto.
- `ReturnValues`: `UPDATED_NEW`

- Si actualiza un elemento existente, `UPDATED_NEW` devuelve solamente los atributos afectados, tal y como aparecen después de la actualización.
- Si actualiza un elemento que no existía (upsert), `UPDATED_NEW` devuelve solamente los atributos actualizados, tal y como aparecen después de la actualización.

DeleteItem

- `ReturnValues`: `ALL_OLD`
  - Si elimina un elemento existente, `ALL_OLD` devuelve el elemento completo tal y como aparecía antes de eliminarlo.
  - Si elimina un elemento que no existía, `ALL_OLD` no devuelve ningún dato.

Operaciones por lotes

Para las aplicaciones que requieren leer o escribir varios elementos, DynamoDB proporciona las operaciones `BatchGetItem` y `BatchWriteItem`. El uso de estas operaciones puede reducir el número de recorridos de ida y vuelta a través de la red entre la aplicación y DynamoDB. Además, DynamoDB lleva a cabo las operaciones de lectura y escritura individuales en paralelo. Las aplicaciones se benefician de este procesamiento en paralelo sin tener que administrar la concurrencia ni los subprocesos.

En esencia, las operaciones por lotes son encapsuladores que incluyen varias solicitudes de lectura o escritura. Por ejemplo, si una solicitud `BatchGetItem` contiene cinco elementos, DynamoDB lleva a cabo cinco operaciones `GetItem`. De igual modo, si una solicitud `BatchWriteItem` contiene dos solicitudes de colocación y cuatro de eliminación, DynamoDB llevará a cabo dos solicitudes `PutItem` y cuatro solicitudes `DeleteItem`.

En general, una operación por lotes no genera un error a no ser que todas las solicitudes del lote generen un error. Por ejemplo, suponga que lleva a cabo una operación `BatchGetItem`, pero que se produce un error en una de las solicitudes `GetItem` individuales del lote. En este caso, `BatchGetItem` devolverá las claves y los datos de la solicitud `GetItem` en la que se ha producido el error. Las demás solicitudes `GetItem` del lote no se ven afectadas.

BatchGetItem

Una sola operación `BatchGetItem` puede contener hasta 100 solicitudes `GetItem` individuales y recuperar hasta 16 MB de datos. Además, una operación `BatchGetItem` puede recuperar elementos de varias tablas.

Example

Recupere dos elementos de la tabla `Thread` usando una expresión de proyección para devolver solo algunos de los atributos.

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json
```

Los argumentos de `--request-items` se almacenan en el archivo `request-items.json`.

```
{
    "Thread": {
        "Keys": [
            {
                "ForumName":{"S": "Amazon DynamoDB"},
                "Subject":{"S": "DynamoDB Thread 1"}
            },
            {
                "ForumName":{"S": "Amazon S3"},
                "Subject":{"S": "S3 Thread 1"}
            }
        ],
        "ProjectionExpression":"ForumName, Subject, LastPostedDateTime, Replies"
    }
}
```

BatchWriteItem

La operación `BatchWriteItem` puede contener hasta 25 solicitudes `PutItem` y `DeleteItem` individuales y puede escribir hasta 16 MB de datos. (El tamaño máximo de un elemento individual es de 400 KB). Además, una operación `BatchWriteItem` puede colocar o eliminar elementos en varias tablas.

> **Note**
>
> `BatchWriteItem` no admite las solicitudes `UpdateItem`.

Example

Escriba dos elementos en la tabla `ProductCatalog`.

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json
```

Los argumentos de `--request-items` se almacenan en el archivo `request-items.json`.

```
{
    "ProductCatalog": [
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "601" },
                    "Description": { "S": "Snowboard" },
                    "QuantityOnHand": { "N": "5" },
                    "Price": { "N": "100" }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": { "N": "602" },
                    "Description": { "S": "Snow shovel" }
                }
            }
        }
    ]
}
```

Contadores atómicos

Puede usar la operación `UpdateItem` para implementar un contador atómico. Se trata de un atributo numérico que se incrementa de forma incondicional y sin interferir con las demás solicitudes

de escritura. Todas las solicitudes de escritura se aplican en el orden en que se reciben. Con un contador atómico, las actualizaciones no son idempotentes. Esto significa que el valor numérico aumenta o disminuye cada vez que se llame a `UpdateItem`. Si el valor de incremento que se usa para actualizar el contador atómico es positivo, se puede producir un recuento excesivo. Si el valor de incremento es negativo, se puede producir un recuento insuficiente.

Es posible utilizar un contador atómico para realizar el seguimiento del número de visitantes de un sitio web. En este caso, la aplicación incrementaría un valor numérico, independientemente del valor actual. En caso de error en la operación `UpdateItem`, la aplicación podría simplemente volver a intentar la operación. Aunque se correría el riesgo de actualizar dos veces el contador, seguramente sería tolerable un pequeño margen de error al alza o a la baja en el número de visitantes del sitio web.

Un contador atómico no sería adecuado en aquellos casos en que no fuese admisible un margen de error al alza o a la baja (por ejemplo, en una aplicación bancaria). En este caso, es más seguro utilizar una actualización condicional en lugar de un contador atómico.

Para obtener más información, consulte Aumento y reducción de atributos numéricos.

Example

En el siguiente ejemplo de la AWS CLI se incrementa el valor de `Price` de un producto en 5. En este ejemplo, se sabía que el elemento existía antes de que se actualizara el contador. Dado que `UpdateItem` no es idempotente, el valor de `Price` aumentará cada vez que se ejecute el código.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id": { "N": "601" }}' \
    --update-expression "SET Price = Price + :incr" \
    --expression-attribute-values '{":incr":{"N":"5"}}' \
    --return-values UPDATED_NEW
```

Escrituras condicionales

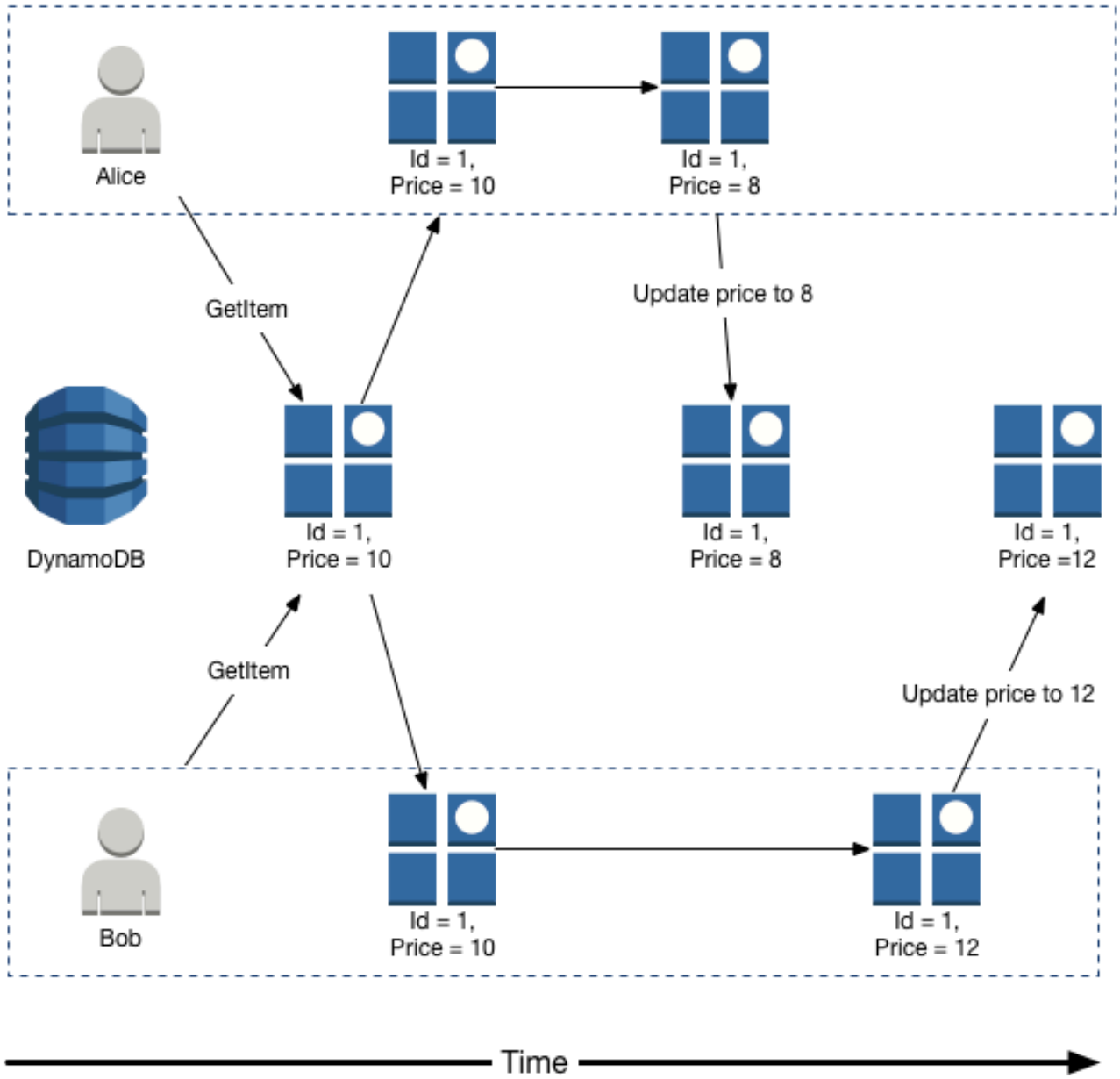
De forma predeterminada, las operaciones de escritura de DynamoDB (`PutItem`, `UpdateItem`, `DeleteItem`) son incondicionales. Cada una de ellas sobrescribirá cualquier elemento existente que tenga la clave principal especificada.

Opcionalmente, DynamoDB admite las escrituras condicionales para estas operaciones. Una escritura condicional solamente se lleva a cabo si los atributos del elemento cumplen una o varias de las condiciones esperadas. En caso contrario, devuelve un error.

En las escrituras condicionales se comprueban las condiciones con la versión actualizada más reciente del elemento. Tenga en cuenta que si el elemento no existía anteriormente o si la operación más reciente que se realizó de forma correcta con ese elemento fue eliminarlo, la escritura condicional no encontrará ningún elemento anterior.

Las escrituras condicionales resultan útiles en muchas situaciones. Por ejemplo, puede ser conveniente que una operación `PutItem` solamente se lleve a cabo si no existe ningún elemento que tenga la misma clave principal. O puede que desee impedir que una operación `UpdateItem` modifique un elemento si uno de sus atributos tiene un valor determinado.

Las escrituras condicionales son útiles en aquellos casos en que varios usuarios intentan modificar el mismo elemento. Fíjese en el siguiente diagrama, en el que dos usuarios (Alice y Bob) trabajan con el mismo elemento de una tabla de DynamoDB.



Suponga que Alice utiliza la AWS CLI para actualizar el atributo `Price` a 8.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --update-expression "SET Price = :newval" \
```

```
    --expression-attribute-values file://expression-attribute-values.json
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo `expression-attribute-values.json`:

```
{
    ":newval":{"N":"8"}
}
```

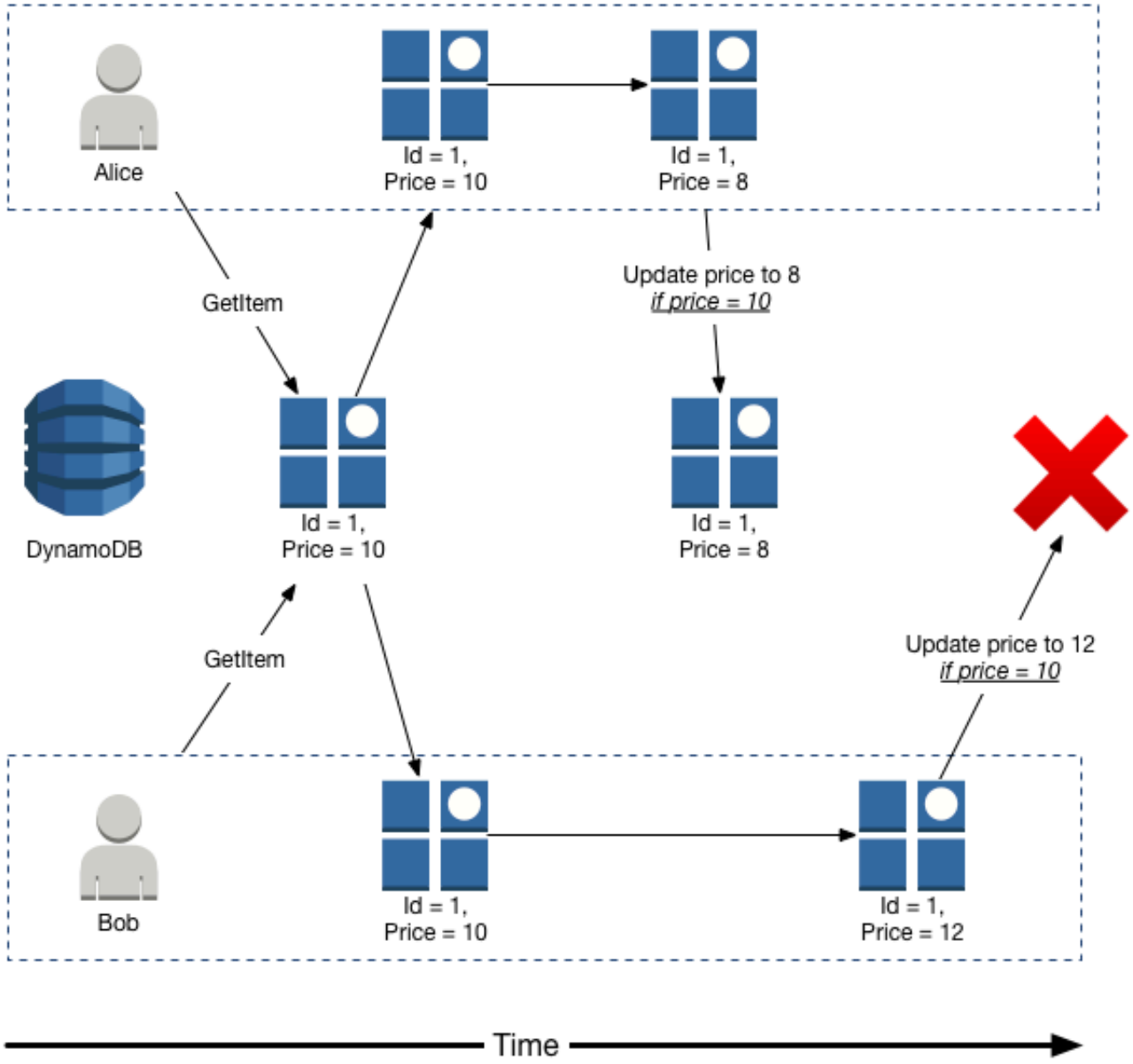
Ahora, supongamos que Bob emite una solicitud `UpdateItem` parecida más adelante, pero cambia el valor de `Price` a 12. Para Bob, el parámetro `--expression-attribute-values` tendrá el siguiente aspecto:

```
{
    ":newval":{"N":"12"}
}
```

Bob la solicitud de Bob se lleva a cabo, pero se pierde la actualización previa de Alice.

Para solicitar una operación `PutItem`, `DeleteItem` o `UpdateItem` condicional, debe especificar una expresión de condición. Una expresión de condición es una cadena que contiene nombres de atributos, operadores condicionales y funciones integradas. La totalidad de expresión debe evaluarse en true. De lo contrario, la operación no se llevará a cabo correctamente.

Ahora, fíjese en el siguiente diagrama, en el que se muestra que el uso de escrituras condicionales impediría que la actualización de Alice se sobrescribiese.



Alice intenta actualizar el valor de `Price` a 8, pero solamente si el valor de `Price` actual es 10.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"1"}}' \
    --update-expression "SET Price = :newval" \
    --condition-expression "Price = :currval" \
```

```
    --expression-attribute-values file://expression-attribute-values.json
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo `expression-attribute-values.json`.

```
{
    ":newval":{"N":"8"},
    ":currval":{"N":"10"}
}
```

La actualización de Alice se lleva a cabo correctamente porque el resultado de evaluar la condición es true.

A continuación, Bob intenta actualizar el valor de `Price` a 12, pero solamente si el valor de `Price` actual es 10. Para Bob, el parámetro `--expression-attribute-values` tendrá el siguiente aspecto:

```
{
    ":newval":{"N":"12"},
    ":currval":{"N":"10"}
}
```

Dado que Alice ha cambiado previamente el valor de `Price` a 8, la expresión de condición se evalúa en false, de modo que la actualización de Bob no se lleva a cabo.

Para obtener más información, consulte Ejemplo de la CLI de expresión de condición de DynamoDB.

Idempotencia de las escrituras condicionales

Las escrituras condicionales pueden ser idempotentes si la comprobación adicional está en el mismo atributo que se va a actualizar. Esto significa que DynamoDB solo realiza una determinada solicitud de escritura si determinados valores de atributo del elemento coinciden con lo que se espera que sean en el momento de la solicitud.

Por ejemplo, suponga que emite una solicitud `UpdateItem` para aumentar el valor de `Price` de un elemento en 3, pero solamente si el valor de `Price` actual es 20. Después de enviar la solicitud, pero antes de recibir su resultado, se produce un error en la red y usted no sabe si la solicitud se ha realizado correctamente. Como esta escritura condicional es idempotente, puede reintentar la misma solicitud `UpdateItem` y DynamoDB actualiza el elemento solamente si el `Price` de actual es 20.

Unidades de capacidad consumidas por las escrituras condicionales

Aunque el resultado de evaluar una expresión `ConditionExpression` sea false durante una escritura condicional, DynamoDB consume capacidad de escritura de la tabla. La cantidad consumida depende del tamaño del elemento existente (o de un mínimo de 1). Por ejemplo, si el tamaño de un elemento existente es de 300 KB y el del nuevo elemento que intenta crear o actualizar es de 310 KB, las unidades de capacidad de escritura consumidas serán 300 si falla la condición y 310 si se realiza correctamente. Si se trata de un elemento nuevo (no existente), las unidades de capacidad de escritura consumidas serán 1 si la condición falla y 310 si la condición se realiza correctamente.

> **Note**
>
> Las operaciones de escritura solo consumen unidades de capacidad de escritura. Nunca consumen unidades de capacidad de lectura.

Una escritura condicional que no se ha llevado a cabo devuelve la excepción `ConditionalCheckFailedException`. Cuando esto ocurre, no se recibe ninguna información en la respuesta sobre la capacidad de escritura que se ha consumido.

Para devolver el número de unidades de capacidad de escritura consumidas durante una escritura condicional, se usa el parámetro `ReturnConsumedCapacity`:

- `TOTAL`: devuelve el número total de unidades de capacidad de escritura consumidas.
- `INDEXES`: devuelve el número total de unidades de capacidad de escritura consumidas, con subtotales para la tabla y todos los índices secundarios que se hayan visto afectados por la operación.
- `NONE`: no devuelve ningún dato de capacidad de escritura consumida. (Esta es la opción predeterminada.)

> **Note**
>
> A diferencia de un índice secundario global, un índice secundario local comparte su capacidad de rendimiento aprovisionada con su tabla. La actividad de lectura y escritura en un índice secundario local consume capacidad de rendimiento aprovisionada de la tabla.

Uso de expresiones en DynamoDB

En Amazon DynamoDB, puede usar expresiones para especificar qué atributos leer de un elemento, escribir datos cuando se cumple una condición, especificar cómo actualizar un elemento, definir consultas y filtrar los resultados de una consulta.

En esta tabla se describen los aspectos gramaticales básicos de las expresiones y los tipos de expresiones disponibles.

Tipo de expresión	Descripción
Expresión de proyección	Una expresión de proyección identifica los atributos que se desean recuperar de un elemento cuando se utilizan operaciones como GetItem, Query o Scan.
Expresión de condición	Una expresión de condición determina qué elementos deben modificarse al utilizar las operaciones PutItem, UpdateItem y DeleteItem.
Expresión de actualización	Una expresión de actualización especifica cómo UpdateItem modificará los atributos de un elemento; por ejemplo, estableciendo un valor escalar o eliminando elementos de una lista o un mapa.
Expresión de condición de clave	Una expresión de condición clave determina qué elementos leerá una consulta de una tabla o índice.
Expresión de filtro	Una expresión de filtro determina qué elementos entre los resultados de consulta se deben devolver al usuario. Todos los demás resultados se descartan.

Para obtener información sobre la sintaxis de expresión e información más detallada sobre cada tipo de expresión, consulte las secciones siguientes.

Temas

- Referencia a atributos de elementos mediante expresiones en DynamoDB
- Nombres de atributos de expresión (Alias) en DynamoDB
- Uso de valores de atributos de expresión en DynamoDB
- Uso de expresiones de proyección en DynamoDB
- Uso de expresiones de actualización en DynamoDB
- Expresiones, operadores y funciones de condición y filtro en DynamoDB
- Ejemplo de la CLI de expresión de condición de DynamoDB

> **Note**
>
> Con el fin de ofrecer compatibilidad retroactiva, DynamoDB también admite parámetros condicionales que no utilizan expresiones. Para obtener más información, consulte Parámetros condicionales de DynamoDB heredados.
> Las nuevas aplicaciones deben utilizar expresiones en lugar de los parámetros heredados.

Referencia a atributos de elementos mediante expresiones en DynamoDB

En esta sección se describe cómo consultar los atributos de los elementos en una expresión en Amazon DynamoDB. Puede utilizar cualquier atributo, aunque se encuentre anidado profundamente en varias listas y mapas.

Temas

- Atributos de nivel superior
- Atributos anidados
- Rutas de documento

Ejemplo de elemento: ProductCatalog

Los ejemplos de esta página utilizan el siguiente elemento de muestra de la tabla de `ProductCatalog`. Esta tabla se describe en . Tablas y datos de ejemplo para usar en DynamoDB.)

```
{
    "Id": 123,
    "Title": "Bicycle 123",
```

```
    "Description": "123 description",
    "BicycleType": "Hybrid",
    "Brand": "Brand-Company C",
    "Price": 500,
    "Color": ["Red", "Black"],
    "ProductCategory": "Bicycle",
    "InStock": true,
    "QuantityOnHand": null,
    "RelatedItems": [
        341,
        472,
        649
    ],
    "Pictures": {
        "FrontView": "http://example.com/products/123_front.jpg",
        "RearView": "http://example.com/products/123_rear.jpg",
        "SideView": "http://example.com/products/123_left_side.jpg"
    },
    "ProductReviews": {
     "FiveStar": [
       "Excellent! Can't recommend it highly enough! Buy it!",
       "Do yourself a favor and buy this."
     ],
     "OneStar": [
       "Terrible product! Do not buy this."
     ]
    },
    "Comment": "This product sells out quickly during the summer",
    "Safety.Warning": "Always wear a helmet"
}
```

Tenga en cuenta lo siguiente:

- El valor de la clave de partición (`Id`) es `123`. No hay clave de ordenación.
- Los tipos de datos de la mayoría de los atributos son escalares, tales como `String`, `Number`, `Boolean` y `Null`.
- Un atributo (`Color`) es de tipo `String Set`.
- Los siguientes atributos tienen tipos de datos de documento:
  - Una lista de `RelatedItems`. Cada componente es un `Id` de un producto relacionado.
  - Mapa de imágenes () `Pictures`. Cada componente es una descripción breve de una imagen, junto con una dirección URL del archivo de imagen correspondiente.

- Mapa de imágenes () `ProductReviews`. Cada componente representa una clasificación y una lista de opiniones correspondientes a esa clasificación. Inicialmente, este mapa se rellena con opiniones de cinco estrellas y una estrella.

Atributos de nivel superior

Se considera que un atributo es de nivel superior si no está integrado en otro atributo. En el elemento `ProductCatalog`, los atributos de nivel superior son:

- `Id`
- `Title`
- `Description`
- `BicycleType`
- `Brand`
- `Price`
- `Color`
- `ProductCategory`
- `InStock`
- `QuantityOnHand`
- `RelatedItems`
- `Pictures`
- `ProductReviews`
- `Comment`
- `Safety.Warning`

Todos estos atributos de nivel superior son escalares, con la salvedad de `Color` (lista), `RelatedItems` (lista), `Pictures` (mapa) y `ProductReviews` (mapa).

Atributos anidados

Se considera que un atributo es anidado si está integrado en otro atributo. Para obtener acceso a un atributo anidado, se utilizan los operadores de desreferenciación:

- `[n]`: para elementos de lista

- `.` (punto): para elementos de mapa

Acceso a los elementos de la lista

El operador de desreferencia de un elemento de la lista es [N], donde n es el número del elemento. Las entradas de lista están basadas en cero; es decir, [0] representa la primera entrada de la lista, [1] representa la segunda, y así sucesivamente. Estos son algunos ejemplos:

- `MyList[0]`
- `AnotherList[12]`
- `ThisList[5][11]`

La entrada `ThisList[5]` es una lista anidada en sí misma. Por consiguiente, `ThisList[5][11]` se refiere al duodécimo componente de esa lista.

El número contenido entre corchetes debe ser un entero no negativo. Por lo tanto, las siguientes expresiones no son válidas:

- `MyList[-1]`
- `MyList[0.4]`

Acceso a los elementos de un mapa

El operador de desreferenciación de una entrada de un mapa es `.` (un punto). Utilice el punto como separador entre las entradas de un mapa:

- `MyMap.nestedField`
- `MyMap.nestedField.deeplyNestedField`

Rutas de documento

En una expresión, se utiliza una ruta de documento para indicar a DynamoDB dónde se encuentra un atributo. En el caso de un atributo de nivel superior, la ruta de documento es el nombre de atributo. En el caso de un atributo anidado, se utilizan operadores de desreferenciación para construir la ruta de documento.

A continuación se indican algunos ejemplos de rutas de documentos. Consulte el elemento mostrado en . Referencia a atributos de elementos mediante expresiones en DynamoDB.)

- Atributo escalar de nivel superior.

  `Description`

- Atributo de lista de nivel superior. (Devuelve la lista completa, no solo algunos de los componentes).

  `RelatedItems`

- Tercera entrada de la lista `RelatedItems`. Recuerde que las entradas de lista se basan en cero.

  `RelatedItems[2]`

- Imagen frontal del producto.

  `Pictures.FrontView`

- Todas las opiniones de cinco estrellas.

  `ProductReviews.FiveStar`

- Primera de las opiniones de cinco estrellas.

  `ProductReviews.FiveStar[0]`

> **Note**
>
> La profundidad máxima de una ruta de documento es 32. Por lo tanto, el número de operadores de desreferenciación de una ruta no puede superar este límite.

Puede utilizar cualquier nombre de atributo en la ruta de un documento siempre que cumpla estos requisitos:

- El primer carácter es `a-z`, `A-Z` o `0-9`
- El segundo carácter (si está presente) es `a-z` o `A-Z`

> **Note**
>
> Si un nombre de atributo no cumple este requisito, tendrá que definir un nombre de atributo de expresión como marcador de posición.

Para obtener más información, consulte Nombres de atributos de expresión (Alias) en DynamoDB.

Nombres de atributos de expresión (Alias) en DynamoDB

Un nombre de atributo de expresión es un alias (o marcador de posición) que se utiliza en una expresión de Amazon DynamoDB en lugar del nombre de atributo real. Un nombre de atributo de expresión debe comenzar por un signo de almohadilla (#) y debe ir seguido de uno o más caracteres alfanuméricos. También se permite el carácter de subrayado (_).

En esta sección se describen varias situaciones en las que debe utilizar nombres de atributos de expresión.

> **Note**
>
> En los ejemplos de esta sección se utiliza la AWS Command Line Interface (AWS CLI).

Temas

- Palabras reservadas
- Nombres de atributo que contienen caracteres especiales
- Atributos anidados
- Hacer referencia repetidamente a nombres de atributos

Palabras reservadas

En algunas ocasiones, es posible que necesite escribir una expresión que contenga un nombre de atributo que entre en conflicto con una palabra reservada de DynamoDB. Para obtener una lista completa de palabras reservadas, consulte . Palabras reservadas en DynamoDB.)

Por ejemplo, el siguiente ejemplo de la AWS CLI no funcionaría correctamente porque `COMMENT` es una palabra reservada,

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Comment"
```

Para solucionar este problema, puede sustituir `Comment` por un nombre de atributo de expresión; por ejemplo, `#c`. El símbolo de almohadilla (#) es obligatorio e indica que se trata de un marcador de posición del nombre de un atributo. Ahora, el ejemplo de la AWS CLI tendría el siguiente aspecto.

```
aws dynamodb get-item \
     --table-name ProductCatalog \
     --key '{"Id":{"N":"123"}}' \
     --projection-expression "#c" \
     --expression-attribute-names '{"#c":"Comment"}'
```

> **Note**
>
> Si un nombre de atributo comienza por un número, contiene un espacio o contiene una palabra reservada, debe utilizar un nombre de atributo de expresión para reemplazar el nombre de ese atributo en la expresión.

Nombres de atributo que contienen caracteres especiales

En una expresión, un punto (".") se interpreta como un carácter separador en una ruta de documento. No obstante, DynamoDB también le permite utilizar un carácter de punto y otros caracteres especiales, como un guion ("-") como parte de un nombre de atributo. En algunos casos, esto puede dar lugar a ambigüedades. A modo de ejemplo, supongamos que desea recuperar el atributo `Safety.Warning` de un elemento `ProductCatalog` (consulte Referencia a atributos de elementos mediante expresiones en DynamoDB).

Supongamos que desea obtener acceso a `Safety.Warning` mediante una expresión de proyección.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "Safety.Warning"
```

DynamoDB podría devolver un resultado vacío, en lugar de la cadena prevista (`"Always wear a helmet"`). El motivo es que DynamoDB interpreta el punto en una expresión como un separador de ruta de documento. En este caso, tiene que definir un nombre de atributo de expresión (por ejemplo, `#sw`) como sustituto de `Safety.Warning`. A continuación, podría utilizar la siguiente expresión de proyección.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#sw" \
    --expression-attribute-names '{"#sw":"Safety.Warning"}'
```

Ahora, DynamoDB devolvería el resultado correcto.

> **Note**
>
> Si un nombre de atributo contiene un punto (".") o un guion ("-"), debe utilizar un nombre de atributo de expresión para reemplazar el nombre de ese atributo en la expresión.

Atributos anidados

Suponga que desea acceder al atributo anidado `ProductReviews.OneStar`. En el nombre de un atributo de expresión, DynamoDB trata el punto (".") como un carácter del nombre del atributo. Para hacer referencia al atributo anidado, defina un nombre de atributo de expresión para cada elemento de la ruta del documento:

- `#pr – ProductReviews`
- `#1star – OneStar`

A continuación, habría que usar `#pr.#1star` para la expresión de proyección.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#pr.#1star"  \
    --expression-attribute-names '{"#pr":"ProductReviews", "#1star":"OneStar"}'
```

Ahora, DynamoDB devolvería el resultado correcto.

Hacer referencia repetidamente a nombres de atributos

Expresión los nombres de atributos de expresión resultan útiles cuando es preciso consultar repetidamente el mismo nombre de atributo. Por ejemplo, tomemos la siguiente expresión para recuperar algunas de las opiniones de un elemento de `ProductCatalog`.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "ProductReviews.FiveStar, ProductReviews.ThreeStar,
 ProductReviews.OneStar"
```

Para que resulte más concisa, puede sustituir `ProductReviews` por un nombre de atributo de expresión, como `#pr`. Ahora, la expresión revisada tendría el siguiente aspecto.

- `#pr.FiveStar, #pr.ThreeStar, #pr.OneStar`

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"123"}}' \
    --projection-expression "#pr.FiveStar, #pr.ThreeStar, #pr.OneStar" \
    --expression-attribute-names '{"#pr":"ProductReviews"}'
```

Si define un nombre de atributo de expresión, debe usarlo de forma coherente en toda la expresión. Además, es importante no omitir el signo #.

Uso de valores de atributos de expresión en DynamoDB

Los valores de los atributos de expresión de Amazon DynamoDB actúan como variables. Reemplazan a los valores reales que se desea comparar, que podrían no conocerse hasta el tiempo de ejecución. Un valor de atributo de expresión debe comenzar por un signo de dos puntos (`:`) y debe ir seguido de uno o más caracteres alfanuméricos.

Por ejemplo, supongamos que desea devolver todos los elementos de `ProductCatalog`que estén disponibles en el color `Black` y tengan un precio de `500` o menos. Podría utilizar una operación `Scan` con una expresión de filtro, como en este ejemplo de la AWS Command Line Interface (AWS CLI).

```
aws dynamodb scan \
    --table-name ProductCatalog \
    --filter-expression "contains(Color, :c) and Price <= :p" \
    --expression-attribute-values file://values.json
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo `values.json`.

```
{
    ":c": { "S": "Black" },
    ":p": { "N": "500" }
}
```

Si define un valor de atributo de expresión, debe usarlo de forma coherente en toda la expresión. Además, es importante no omitir el signo `:`.

Los valores de atributos de expresión se usan con expresiones de condición de clave, expresiones de condición, expresiones de actualización y expresiones de filtro.

Uso de expresiones de proyección en DynamoDB

Para leer datos de una tabla, se utilizan operaciones tales como `GetItem`, `Query` o `Scan`. De forma predeterminada, Amazon DynamoDB devuelve todos los atributos de los elementos. Si desea obtener solo uno en lugar de todos ellos, debe usar una expresión de proyección.

Una expresión de proyección es una cadena que identifica los atributos que se desea. Para recuperar un solo atributo, especifique su nombre. Si desea obtener varios atributos, separe sus nombres mediante comas.

A continuación se muestran ejemplos de expresiones de proyección basadas en el elemento `ProductCatalog` de Referencia a atributos de elementos mediante expresiones en DynamoDB:

- Un solo atributo de nivel superior.

  ```
  Title
  ```

- Tres atributos de nivel superior. DynamoDB recupera todo el conjunto `Color`.

  ```
  Title, Price, Color
  ```

- Cuatro atributos de nivel superior. DynamoDB devolverá todo el contenido de `RelatedItems` y `ProductReviews`.

  ```
  Title, Description, RelatedItems, ProductReviews
  ```

> **Note**
>
> La expresión de proyección no modifica el consumo de rendimiento aprovisionado. DynamoDB determina las unidades de capacidad consumidas según el tamaño de los elementos y no según la cantidad de datos que se devuelven a una aplicación.

Palabras reservadas y caracteres especiales

DynamoDB tiene palabras reservadas y caracteres especiales. DynamoDB le permite utilizar estas palabras reservadas y caracteres especiales para nombres, pero le recomendamos que evite hacerlo, porque tendría que usar alias para ellos cada vez que utilizase estos nombres en una expresión. Para ver una lista completa, consulte Palabras reservadas en DynamoDB.

Deberá usar los nombres de los atributos de la expresión en lugar del nombre real si:

- El nombre del atributo está en la lista de palabras reservadas en DynamoDB.
- El nombre del atributo no cumple el requisito de que el primer carácter sea `a-z` o `A-Z` y que el segundo carácter (si lo hay) sea `a-Z`, `A-Z` o `0-9`.
- El nombre del atributo contiene un signo # (hash) o : (dos puntos).

En el ejemplo de la AWS CLI siguiente se muestra cómo usar una expresión de proyección con una operación `GetItem`. La expresión de proyección recupera un atributo escalar de nivel superior (`Description`), la primera entrada de una lista (`RelatedItems[0]`) y una lista anidada en un mapa (`ProductReviews.FiveStar`).

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '"Id": { "N": "123" } \
    --projection-expression "Description, RelatedItems[0], ProductReviews.FiveStar"
```

Para este ejemplo, se devolvería el siguiente código JSON.

```
{
    "Item": {
        "Description": {
            "S": "123 description"
        },
```

```
        "ProductReviews": {
            "M": {
                "FiveStar": {
                    "L": [
                        {
                            "S": "Excellent! Can't recommend it highly enough! Buy it!"
                        },
                        {
                            "S": "Do yourself a favor and buy this."
                        }
                    ]
                }
            }
        },
        "RelatedItems": {
            "L": [
                {
                    "N": "341"
                }
            ]
        }
    }
}
```

Uso de expresiones de actualización en DynamoDB

La acción `UpdateItem` actualiza un elemento existente o agrega uno nuevo a la tabla, si no existe ya. Es preciso proporcionar la clave del elemento que se desea actualizar. Asimismo, debe proporcionar una expresión de actualización que indique los atributos que se van a modificar y los valores que se les asignarán.

Una expresión de actualización especifica cómo `UpdateItem` modificará los atributos de un elemento; por ejemplo, estableciendo un valor escalar o eliminando elementos de una lista o un mapa.

A continuación se muestra un resumen de la sintaxis de las expresiones de actualización.

```
update-expression ::=
    [ SET action [, action] ... ]
    [ REMOVE action [, action] ...]
    [ ADD action [, action] ... ]
    [ DELETE action [, action] ...]
```

Una expresión de actualización consta de una o varias cláusulas. Cada cláusula comienza con una palabra clave `SET`, `REMOVE`, `ADD` o `DELETE`. Puede incluir cualquiera de estas cláusulas en una expresión de actualización, en cualquier orden. Sin embargo, cada palabra clave de acción solo puede aparecer una vez.

Cada cláusula contiene una o más acciones, separadas por comas. Cada acción representa una modificación de datos.

Los ejemplos que aparecen en esta sección se basan en el elemento `ProductCatalog` que se muestra en Uso de expresiones de proyección en DynamoDB.

En los temas siguientes se describen algunos casos de uso diferentes de la acción `SET`.

Temas

- SET: modificación o adición de atributos de elemento
- REMOVE: eliminación de atributos de un elemento
- ADD: actualización de números y conjuntos
- DELETE: eliminación de elementos de un conjunto
- Uso de varias expresiones de actualización

SET: modificación o adición de atributos de elemento

Utilice la acción `SET` en una expresión de actualización para agregar uno o varios atributos a un elemento. Si cualquiera de estos atributos ya existe, se sobrescribirá con los nuevos valores. Si desea evitar sobrescribir un atributo existente, puede utilizar `SET` con la función `if_not_exists`. La función `if_not_exists` es específica de la acción `SET` y solamente se puede utilizar en una expresión de actualización.

Cuando se utiliza `SET` para actualizar una entrada de lista, el contenido de esa entrada se sustituye por los nuevos datos que ha especificado. Si el componente no existe, `SET` adjunta el nuevo componente al final de la lista.

Si agrega varias entradas en una misma operación `SET`, las entradas se ordenan según su número.

También puede utilizar `SET` para sumar o restar un valor de un atributo de tipo `Number`. Para llevar a cabo varias acciones `SET`, debe separarlas por comas.

En el resumen de sintaxis siguiente:

- El componente *path* es la ruta de documento del elemento.
- Un componente *operand* puede ser una ruta de documento a un elemento o una función.

```
set-action ::=
    path = value

value ::=
    operand
    | operand '+' operand
    | operand '-' operand

operand ::=
    path | function

function ::=
    if_not_exists (path, value)
```

Si el elemento no contiene un atributo en la ruta especificada, `if_not_exists` se evalúa en `value`. De lo contrario, se evalúa como `path`.

La siguiente operación `PutItem` crea un elemento de muestra al que se refieren los ejemplos.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json
```

Los argumentos de `--item` se almacenan en el archivo `item.json`. Para simplificar, se utilizan tan solo algunos de los atributos de elementos.

```
{
    "Id": {"N": "789"},
    "ProductCategory": {"S": "Home Improvement"},
    "Price": {"N": "52"},
    "InStock": {"BOOL": true},
    "Brand": {"S": "Acme"}
}
```

Temas

- Modificación de atributos

- Adición de listas y mapas
- Adición de elementos a una lista
- Adición de atributos de mapa anidados
- Aumento y reducción de atributos numéricos
- Adición de elementos a una lista
- Cómo evitar sobrescribir un atributo existente

Modificación de atributos

Example

Actualice los atributos `ProductCategory` y `Price`.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET ProductCategory = :c, Price = :p" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo `values.json`.

```
{
    ":c": { "S": "Hardware" },
    ":p": { "N": "60" }
}
```

> **Note**
>
> En la operación `UpdateItem`, `--return-values ALL_NEW` hace que DynamoDB devuelva el elemento tal y como aparece después de la actualización.

Adición de listas y mapas

Example

Agregue una lista y un mapa nuevos.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems = :ri, ProductReviews = :pr" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo `values.json`.

```
{
    ":ri": {
        "L": [
            { "S": "Hammer" }
        ]
    },
    ":pr": {
        "M": {
            "FiveStar": {
                "L": [
                    { "S": "Best product ever!" }
                ]
            }
        }
    }
}
```

Adición de elementos a una lista

Example

Agregue un nuevo atributo a la lista `RelatedItems`. Recuerde que las entradas de lista están basadas en cero; es decir, [0] representa la primera entrada de la lista, [1] representa la segunda, y así sucesivamente.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems[1] = :ri" \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo `values.json`.

```
{
    ":ri": { "S": "Nails" }
}
```

> **Note**
>
> Cuando se utiliza SET para actualizar una entrada de lista, el contenido de esa entrada se sustituye por los nuevos datos que ha especificado. Si el componente no existe, SET adjunta el nuevo componente al final de la lista.
> Si agrega varias entradas en una misma operación SET, las entradas se ordenan según su número.

Adición de atributos de mapa anidados

Example

Agregue algunos atributos de mapa anidados.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET #pr.#5star[1] = :r5, #pr.#3star = :r3" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_NEW
```

Los argumentos de `--expression-attribute-names` se almacenan en el archivo `names.json`.

```
{
    "#pr": "ProductReviews",
    "#5star": "FiveStar",
    "#3star": "ThreeStar"
}
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo `values.json`.

```
{
    ":r5": { "S": "Very happy with my purchase" },
    ":r3": {
        "L": [
            { "S": "Just OK - not that great" }
        ]
    }
}
```

Aumento y reducción de atributos numéricos

Puede sumar o restar un valor a un atributo numérico. Para ello, se utilizan los operadores + (más) y - (menos).

Example

Reduzca el `Price` de un elemento.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = Price - :p" \
    --expression-attribute-values '{":p": {"N":"15"}}' \
    --return-values ALL_NEW
```

Para aumentar el valor de `Price`, se utiliza el operador + en la expresión de actualización.

Adición de elementos a una lista

Puede agregar componentes al final de una lista. Para ello, se utiliza `SET` con la función `list_append`. (El nombre de la función distingue entre mayúsculas y minúsculas). La función `list_append` es específica de la acción `SET` y solamente se puede utilizar en una expresión de actualización. La sintaxis es la siguiente.

- `list_append (`*`list1`*`, `*`list2`*`)`

La función toma dos listas como entrada y anexa todos los elementos de *`list2`* a *`list1`*.

Example

En Adición de elementos a una lista, ha creado la lista `RelatedItems` y ha incluido en ella dos componentes: `Hammer` y `Nails`. Ahora, va a agregar dos componentes más al final de `RelatedItems`.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET #ri = list_append(#ri, :vals)" \
    --expression-attribute-names '{"#ri": "RelatedItems"}' \
    --expression-attribute-values file://values.json  \
    --return-values ALL_NEW
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo `values.json`.

```
{
    ":vals": {
        "L": [
            { "S": "Screwdriver" },
            {"S": "Hacksaw" }
        ]
    }
}
```

Por último, vamos a agregar un componente más al principio de `RelatedItems`. Para ello, invierta el orden de los componentes de `list_append`. (Recuerde que `list_append` toma dos listas como información de entrada y agrega la segunda lista a la primera).

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET #ri = list_append(:vals, #ri)" \
    --expression-attribute-names '{"#ri": "RelatedItems"}' \
    --expression-attribute-values '{":vals": {"L": [ { "S": "Chisel" }]}}' \
    --return-values ALL_NEW
```

Ahora, el atributo `RelatedItems` resultante contiene cinco entradas, el siguiente orden: `Chisel`, `Hammer`, `Nails`, `Screwdriver` y `Hacksaw`.

Cómo evitar sobrescribir un atributo existente

Example

Establezca el valor de `Price` de un elemento, pero solo si este no tiene ya un atributo `Price`. (Si `Price` ya existe, no sucede nada).

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = if_not_exists(Price, :p)" \
    --expression-attribute-values '{":p": {"N": "100"}}' \
    --return-values ALL_NEW
```

REMOVE: eliminación de atributos de un elemento

Utilice la acción `REMOVE` en una expresión de actualización para eliminar uno o varios atributos de un elemento en Amazon DynamoDB. Para llevar a cabo varias acciones `REMOVE`, debe separarlas por comas.

A continuación se muestra un resumen de la sintaxis de `REMOVE` en una expresión de actualización. El único operando es la ruta de documento del atributo que se desea eliminar.

```
remove-action ::=
    path
```

Example

Elimine algunos atributos de un elemento. (Si el atributo no existe, no sucede nada).

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE Brand, InStock, QuantityOnHand" \
    --return-values ALL_NEW
```

Eliminación de elementos de una lista

Puede utilizar `REMOVE` para eliminar entradas individuales de una lista.

Example

En Adición de elementos a una lista, ha modificado un atributo de lista (`RelatedItems`) para que contenga cinco componentes:

- `[0]—Chisel`
- `[1]—Hammer`
- `[2]—Nails`
- `[3]—Screwdriver`
- `[4]—Hacksaw`

En el siguiente ejemplo de la AWS Command Line Interface (AWS CLI) se eliminan `Hammer` y `Nails` de la lista.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "REMOVE RelatedItems[1], RelatedItems[2]" \
    --return-values ALL_NEW
```

Después de eliminar `Hammer` y `Nails`, los componentes restantes se reordenan. Ahora, la lista contiene lo siguiente:

- `[0]—Chisel`
- `[1]—Screwdriver`
- `[2]—Hacksaw`

ADD: actualización de números y conjuntos

> **Note**
>
> En general, recomendamos utilizar `SET` en lugar de `ADD`.

Utilice la acción `ADD` en una expresión de actualización para agregar un nuevo atributo y sus valores a un elemento.

Si el atributo ya existe, el comportamiento de `ADD` depende del tipo de datos del atributo:

- Si el atributo es un número y el valor que se agrega también es un número, entonces el valor se suma matemáticamente al atributo existente. (Si el valor es un número negativo, entonces se resta del atributo existente).
- Si el atributo es un conjunto y el valor que se agrega también es un conjunto, entonces el valor se agrega al conjunto existente.

> **Note**
>
> La acción `ADD` solo es compatible con los tipos de datos Number y Set.

Para llevar a cabo varias acciones `ADD`, debe separarlas por comas.

En el resumen de sintaxis siguiente:

- El componente *`path`* es la ruta de documento de un atributo. El tipo de datos del atributo debe ser `Number` o Set.
- El componente *`value`* es un número que se desea agregar al atributo (si el tipo de datos es `Number`) o un conjunto que se desea agregar al atributo (si el tipo de datos es Set).

```
add-action ::=
    path value
```

En los temas siguientes se describen algunos casos de uso diferentes de la acción `ADD`.

Temas

- Adición de un número
- Adición de elementos a un conjunto

Adición de un número

Supongamos que el atributo `QuantityOnHand` no existe. En el siguiente ejemplo de la AWS CLI, `QuantityOnHand` se establece en 5.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
```

```
    --update-expression "ADD QuantityOnHand :q" \
    --expression-attribute-values '{":q": {"N": "5"}}' \
    --return-values ALL_NEW
```

Ahora que `QuantityOnHand` ya existe, puede volver a ejecutar el ejemplo para incrementar `QuantityOnHand` en 5 cada vez.

Adición de elementos a un conjunto

Supongamos que el atributo `Color` no existe. En el siguiente ejemplo de la AWS CLI, `Color` se establece en un conjunto de cadenas que contiene dos componentes.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD Color :c" \
    --expression-attribute-values '{":c": {"SS":["Orange", "Purple"]}}' \
    --return-values ALL_NEW
```

Ahora que `Color` ya existe, podemos agregarle más componentes.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "ADD Color :c" \
    --expression-attribute-values '{":c": {"SS":["Yellow", "Green", "Blue"]}}' \
    --return-values ALL_NEW
```

DELETE: eliminación de elementos de un conjunto

> **Important**
>
> La acción `DELETE` solo es compatible con tipos de datos `Set`.

Utilice la acción `DELETE` en una expresión de actualización para eliminar una o varias entradas de un conjunto. Para llevar a cabo varias acciones `DELETE`, debe separarlas por comas.

En el resumen de sintaxis siguiente:

- El componente *path* es la ruta de documento de un atributo. El tipo de datos del atributo debe ser Set.
- *Subset* representa uno o varios componentes que se van a eliminar de *path*. Para *subset* debe especificar un tipo de datos Set.

```
delete-action ::=
    path subset
```

Example

En Adición de elementos a un conjunto, crea el conjunto de cadenas `Color`. En este ejemplo se eliminan algunos de los componentes de ese conjunto.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "DELETE Color :p" \
    --expression-attribute-values '{":p": {"SS": ["Yellow", "Purple"]}}' \
    --return-values ALL_NEW
```

Uso de varias expresiones de actualización

Puede utilizar varias expresiones de actualización en una instrucción.

Example

Si desea modificar el valor de un atributo y eliminar por completo otro, podría utilizar una acción SET y otra REMOVE en una sola instrucción. Esta operación reduciría el valor de `Price` a 15 a la vez que eliminaría el atributo `InStock` del elemento.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET Price = Price - :p REMOVE InStock" \
    --expression-attribute-values '{":p": {"N":"15"}}' \
    --return-values ALL_NEW
```

Example

Si desea agregar un elemento a una lista al mismo tiempo que cambia el valor de otro atributo, podría utilizar dos acciones SET en una sola instrucción. Esta operación agregaría "Nails" al atributo de la lista `RelatedItems` y también establecería el valor de `Price` a 21.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"789"}}' \
    --update-expression "SET RelatedItems[1] = :newValue, Price = :newPrice" \
    --expression-attribute-values '{":newValue": {"S":"Nails"}, ":newPrice":
 {"N":"21"}}'  \
    --return-values ALL_NEW
```

Expresiones, operadores y funciones de condición y filtro en DynamoDB

Para manipular datos en una tabla de DynamoDB, se usan las operaciones `PutItem`, `UpdateItem` y `DeleteItem`. Para estas operaciones de manipulación de datos, puede especificar una expresión de condición con el fin de determinar qué elementos deben modificarse. Si la expresión de condición se evalúa en true, entonces la operación se realiza correctamente. De lo contrario, la operación no se llevará a cabo correctamente.

En esta sección se describen las funciones y las palabras clave integradas para escribir expresiones de filtro y condición en Amazon DynamoDB. Para obtener más información sobre las funciones y la programación con DynamoDB, consulte Programación con DynamoDB y los SDK de AWS y la Referencia de la API de DynamoDB.

Temas

- Sintaxis de las expresiones de filtro y condición
- Realización de comparaciones
- Funciones
- Evaluaciones lógicas
- Paréntesis
- Precedencia en las condiciones

Sintaxis de las expresiones de filtro y condición

En el siguiente resumen de sintaxis, el componente *operand* puede ser uno de los siguientes:

- Un nombre de atributo de nivel superior, como por ejemplo `Id`, `Title`, `Description` o `ProductCategory`
- Una ruta de documento que hace referencia a un atributo anidado

```
condition-expression ::=
      operand comparator operand
    | operand BETWEEN operand AND operand
    | operand IN ( operand (',' operand (, ...) ))
    | function
    | condition AND condition
    | condition OR condition
    | NOT condition
    | ( condition )

comparator ::=
    =
    | <>
    | <
    | <=
    | >
    | >=

function ::=
    attribute_exists (path)
    | attribute_not_exists (path)
    | attribute_type (path, type)
    | begins_with (path, substr)
    | contains (path, operand)
    | size (path)
```

Realización de comparaciones

Utilice estos comparadores para comparar un operando con un solo valor:

- *a* = *b*: es true si *a* es igual que *b*.
- *a* <> *b*: es true si *a* es distinto de *b*.
- *a* < *b*: es true si *a* es menor que *b*.
- *a* <= *b*: es true si *a* es menor o igual que *b*.
- *a* > *b*: es true si *a* es mayor que *b*.

- *a* >= *b*: es true si *a* es mayor o igual que *b*.

Use las palabras clave BETWEEN e IN para comparar un operando con un rango o una lista de valores:

- *a* BETWEEN *b* AND *c*: es true si *a* es mayor o igual que *b* y menor o igual que *c*.
- *a* IN (*b*, *c*, *d*) : es true si *a* es igual a cualquiera de los valores de la lista; por ejemplo, en este caso, a *b*, *c* o *d*. La lista puede contener hasta 100 valores separadas por comas.

Funciones

Utilice las siguientes funciones para determinar si un atributo existe en un elemento o evaluar el valor de un atributo. Los nombres de estas funciones distinguen entre mayúsculas y minúsculas. En el caso de los atributos anidados, debe proporcionar su ruta de documento completa.

Función	Descripción
attribute_exists (*path*)	Es true si el elemento contiene el atributo especificado por path. Ejemplo: Comprobación de si un elemento de la tabla Product tiene una imagen de vista lateral. • attribute_exists (#Pictures.#SideView)
attribute_not_exists (*path*)	Es true si el atributo especificado en path no está presente en el elemento. Ejemplo: Comprobación de si un elemento tiene el atributo Manufacturer . • attribute_not_exists (Manufacturer)

Función	Descripción
`attribute_type ( path, type)`	Es true si el atributo de la ruta especificada es de un tipo de datos determinado. El parámetro `type` debe ser uno de los siguientes: • `S`: String • `SS`: conjunto de cadenas • `N`: Number • `NS`: conjunto de números • `B`: Binary • `BS`: conjunto de binarios • `BOOL`: Boolean • `NULL`: Null • `L`: List • `M`: Map Debe utilizar un valor de atributo de expresión para el parámetro `type`. Ejemplo: Comprobación de si el atributo `QuantityOnHand` es del tipo List. En este ejemplo, `:v_sub` es un marcador de posición para la `L`. • `attribute_type (ProductReviews.FiveStar, :v_sub)`

Función	Descripción
	Debe utilizar un valor de atributo de expresión para el parámetro `type`.
`begins_with ( path, substr)`	Es true si el atributo especificado por `path` comienza por una subcadena determinada. Ejemplo: Comprobación de si los primeros caracteres de la URL de la imagen de vista frontal son `http://`. • `begins_with (Pictures.FrontView, :v_sub)` El valor de atributo de expresión `:v_sub` es un marcador de posición para `http://`.

Función	Descripción
`contains (path, operand)`	Es true si el atributo especificado por `path` es uno de los siguientes: • Un valor de tipo `String` que contiene una determinada subcadena. • Un valor de tipo `Set` que contiene un componente determinado perteneciente al conjunto. • Un valor de tipo `List` que contiene un componente determinado perteneciente a la lista. Si el atributo especificado por `path` es un `String`, el `operand` debe ser un `String`. Si el atributo especificado por `path` es un `Set`, el `operand` debe ser el tipo de elemento del conjunto. La ruta y el operando deben ser distintos. Es decir, `contains (a, a)` devuelve un error. Ejemplo: Comprobación de si el atributo `Brand` contiene la subcadena `Company`. • `contains (Brand, :v_sub)` El valor de atributo de expresión `:v_sub` es un marcador de posición para `Company`. Ejemplo: Comprobación de si el producto está disponible en color rojo.

Función	Descripción
	• `contains (Color, :v_sub)` El valor de atributo de expresión `:v_sub` es un marcador de posición para `Red`.

Función	Descripción
`size (path)`	Devuelve un número que representa el tamaño de un atributo. A continuación se muestran los tipos de datos válidos para usarlos con `size`. Si el atributo es de tipo `String`, `size` devuelve la longitud de la cadena. Ejemplo: Comprobación de si la cadena `Brand` es menor o igual que 20 caracteres. El valor de atributo de expresión `:v_sub` es un marcador de posición para `20`. • `size (Brand) <= :v_sub` Si el atributo es de tipo `Binary`, `size` devuelve el número de bytes del valor del atributo. Ejemplo: supongamos que el elemento `ProductCatalog` tiene un atributo binario denominado `VideoClip` que contiene vídeo breve sobre el uso del producto. En la siguiente expresión se comprueba si `VideoClip` supera los 64 000 bytes. El valor de atributo de expresión `:v_sub` es un marcador de posición para `64000`. • `size(VideoClip) > :v_sub` Si el tipo de datos del atributo es `Set`, `size` devuelve el número de componentes del conjunto.

Función	Descripción
	Ejemplo: Comprobación de si el producto está disponible en más de un color. El valor de atributo de expresión `:v_sub` es un marcador de posición para `1`. • `size (Color) < :v_sub` Si el atributo es de tipo `List` o `Map`, `size` devuelve el número de componentes secundarios. Ejemplo: Comprobación de si el número de opiniones `OneStar` ha superado un umbral determinado. El valor de atributo de expresión `:v_sub` es un marcador de posición para `3`. • `size(ProductReviews.OneStar) > :v_sub`

Evaluaciones lógicas

Utilice las palabras clave `AND`, `OR` y `NOT` para llevar a cabo evaluaciones lógicas. En la lista siguiente, *a* y *b* representan las condiciones que se van a evaluar.

- *a* `AND` *b*: es true si *a* y *b* son true.
- *a* `OR` *b*: es true si *a* o *b* (o ambas) son true.
- `NOT` *a*: es true si *a* es false. False si *a* es true.

A continuación se muestra un ejemplo de código de AND en una operación.

```
dynamodb-local (*)> select * from exprtest where a > 3 and a < 5;
```

Paréntesis

Los paréntesis se utilizan para cambiar la preferencia de una evaluación lógica. Por ejemplo, supongamos que las condiciones *a* y *b* son true y que la condición *c* es false. La siguiente expresión se evalúa en true:

- `a OR b AND c`

Sin embargo, si se incluye una condición entre paréntesis, esta se evalúa antes. Por ejemplo, lo siguiente se evalúa en false:

- `(a OR b) AND c`

> **Note**
>
> En una expresión se pueden utilizar paréntesis anidados. En este caso, se evalúan primero los más internos.

A continuación se muestra un ejemplo de código con paréntesis en una evaluación lógica.

```
dynamodb-local (*)> select * from exprtest where attribute_type(b, string)
or ( a = 5 and c = “coffee”);
```

Precedencia en las condiciones

DynamoDB evalúa las condiciones de izquierda a derecha aplicando las siguientes normas de prioridad:

- `= <> < <= > >=`
- `IN`
- `BETWEEN`
- `attribute_exists attribute_not_exists begins_with contains`
- Paréntesis
- `NOT`
- `AND`

- OR

Ejemplo de la CLI de expresión de condición de DynamoDB

A continuación se muestran algunos ejemplos de uso de expresiones de condición en la AWS Command Line Interface (AWS CLI). Estos ejemplos se basan en la tabla `ProductCatalog`, especificada en Referencia a atributos de elementos mediante expresiones en DynamoDB. La clave de partición de esta tabla es `Id` y no tiene clave de ordenación. La siguiente operación `PutItem` crea un elemento de muestra `ProductCatalog` al que se refieren los ejemplos.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json
```

Los argumentos de `--item` se almacenan en el archivo `item.json`. Para simplificar, se utilizan tan solo algunos de los atributos de elementos.

```
{
    "Id": {"N": "456" },
    "ProductCategory": {"S": "Sporting Goods" },
    "Price": {"N": "650" }
}
```

Temas

- PUT condicional
- Eliminaciones condicionales
- Actualizaciones condicionales
- Ejemplos de expresiones condicionales

PUT condicional

La operación `PutItem` sobrescribe un elemento que tenga la misma clave principal (si existe). Si desea evitar que esto suceda, utilice una expresión de condición. Esto permite que la escritura se lleve a cabo solo si el elemento en cuestión ya no tiene la misma clave principal.

En el siguiente ejemplo se utiliza `attribute_not_exists()` para comprobar si la clave principal existe en la tabla antes de intentar la operación de escritura.

> **Note**
>
> Si la clave principal consta de una clave de partición (pk) y una clave de clasificación (sk), el parámetro comprobará si `attribute_not_exists(pk)` Y `attribute_not_exists(sk)` se evalúan como verdadero o falso como una instrucción completa antes de intentar la operación de escritura.

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item file://item.json \
    --condition-expression "attribute_not_exists(Id)"
```

Si la expresión de condición se evalúa en false (falso), DynamoDB devuelve el siguiente mensaje de error: The conditional request failed (Se produjo un error en la consulta condicional).

> **Note**
>
> Para obtener más información sobre `attribute_not_exists` y otras funciones, consulte Expresiones, operadores y funciones de condición y filtro en DynamoDB.

Eliminaciones condicionales

Para realizar una eliminación condicional, se usa una operación `DeleteItem` con una expresión de condición. La expresión de condición debe evaluarse en true para que la operación se lleve a cabo correctamente; de lo contrario, se produce un error.

Tenga en cuenta el elemento definido anteriormente.

Suponga que desea eliminar el elemento, pero solo en las siguientes condiciones:

- El valor de `ProductCategory` es "Sporting Goods" o "Gardening Supplies".
- El valor de `Price` está comprendido entre 500 y 600.

En el siguiente ejemplo se intenta eliminar el elemento.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
```

```
    --key '{"Id":{"N":"456"}}' \
    --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (Price between :lo
 and :hi)" \
    --expression-attribute-values file://values.json
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo `values.json`.

```
{
    ":cat1": {"S": "Sporting Goods"},
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

> **Note**
>
> En la expresión de condición, `:` (signo de dos puntos) indica un valor de atributo de expresión (un marcador de posición del valor real). Para obtener más información, consulte Uso de valores de atributos de expresión en DynamoDB.
> Para obtener más información sobre `IN`, `AND` y otras palabras clave, consulte Expresiones, operadores y funciones de condición y filtro en DynamoDB.

En este ejemplo, la comparación de `ProductCategory` se evalúa en true, pero la comparación de `Price` se evalúa en false. Esto hace que la expresión de condición se evalúe en false y, por consiguiente, la operación `DeleteItem` no se lleva a cabo.

Actualizaciones condicionales

Para realizar una actualización condicional, se usa una operación `UpdateItem` con una expresión de condición. La expresión de condición debe evaluarse en true para que la operación se lleve a cabo correctamente; de lo contrario, se produce un error.

> **Note**
>
> `UpdateItem` también admite las expresiones de actualización, donde especifica las modificaciones que se desea aplicar a un elemento. Para obtener más información, consulte Uso de expresiones de actualización en DynamoDB.

Supongamos que ha comenzado por el elemento definido anteriormente.

En el ejemplo siguiente se realiza una operación `UpdateItem`. Se intenta reducir el valor de `Price` de un producto en 75, pero la expresión de condición impide la actualización si el valor de `Price` actual es menor o igual que 500.

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --update-expression "SET Price = Price - :discount" \
    --condition-expression "Price > :limit" \
    --expression-attribute-values file://values.json
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo `values.json`.

```
{
    ":discount": { "N": "75"},
    ":limit": {"N": "500"}
}
```

Si el valor inicial de `Price` es 650, la operación `UpdateItem` reduce el `Price` a 575. Si ejecuta la operación `UpdateItem` de nuevo, el valor de `Price` se reduce a 500. Si se ejecuta una tercera vez, la expresión de condición se evalúa en false y la actualización no se lleva a cabo.

> **Note**
>
> En la expresión de condición, `:` (signo de dos puntos) indica un valor de atributo de expresión (un marcador de posición del valor real). Para obtener más información, consulte Uso de valores de atributos de expresión en DynamoDB.
> Para obtener más información sobre ">" y otros operadores, consulte Expresiones, operadores y funciones de condición y filtro en DynamoDB.

Ejemplos de expresiones condicionales

Para obtener más información acerca de las funciones utilizadas en los ejemplos siguientes, consulte Expresiones, operadores y funciones de condición y filtro en DynamoDB. Si desea obtener más información sobre cómo especificar distintos tipos de atributo en una expresión, consulte Referencia a atributos de elementos mediante expresiones en DynamoDB.

Comprobación de los atributos de un elemento

Puede comprobar la existencia (o inexistencia) de cualquier atributo. Si la expresión de condición se evalúa en true, entonces la operación se realiza correctamente; de lo contrario, produce un error.

En el siguiente ejemplo se utiliza `attribute_not_exists` para eliminar un producto únicamente si no tiene el atributo `Price`.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_not_exists(Price)"
```

DynamoDB también proporciona un función `attribute_exists`. En el siguiente ejemplo se elimina un producto únicamente si ha recibido opiniones negativas.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_exists(ProductReviews.OneStar)"
```

Comprobación del tipo de atributo

Puede comprobar el tipo de datos de un valor de atributo mediante la función `attribute_type`. Si la expresión de condición se evalúa en true, entonces la operación se realiza correctamente; de lo contrario, produce un error.

En el ejemplo siguiente se utiliza `attribute_type` para eliminar un producto sólo si tiene un atributo `Color` de tipo String Set.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "attribute_type(Color, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo expression-attribute-values.json.

```
{
    ":v_sub":{"S":"SS"}
```

```
}
```

Comprobación del valor inicial de cadena

Puede comprobar si un valor de atributo String comienza con una subcadena determinada mediante la función `begins_with`. Si la expresión de condición se evalúa en true, entonces la operación se realiza correctamente; de lo contrario, produce un error.

En el ejemplo siguiente se utiliza `begins_with` para eliminar un producto solo si el elemento `FrontView` del mapa `Pictures` comienza con un valor específico.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "begins_with(Pictures.FrontView, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo expression-attribute-values.json.

```
{
    ":v_sub":{"S":"http://"}
}
```

Comprobación de un elemento en un conjunto

Puede buscar un elemento en un conjunto o buscar una subcadena dentro de una cadena mediante el uso de la función `contains`. Si la expresión de condición se evalúa en true, entonces la operación se realiza correctamente; de lo contrario, produce un error.

En el ejemplo siguiente se utiliza `contains` para eliminar un producto sólo si el conjunto de cadenas `Color` tiene un elemento con un valor específico.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "contains(Color, :v_sub)" \
    --expression-attribute-values file://expression-attribute-values.json
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo expression-attribute-values.json.

```
{
    ":v_sub":{"S":"Red"}
}
```

Comprobación del tamaño de un valor de atributo

Puede comprobar el tamaño de un valor de atributo mediante la función `size`. Si la expresión de condición se evalúa en true, entonces la operación se realiza correctamente; de lo contrario, produce un error.

En el ejemplo siguiente se utiliza `size` para eliminar un producto sólo si el tamaño del atributo Binary `VideoClip` es mayor de `64000` bytes.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "456"}}' \
    --condition-expression "size(VideoClip) > :v_sub" \
    --expression-attribute-values file://expression-attribute-values.json
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo expression-attribute-values.json.

```
{
    ":v_sub":{"N":"64000"}
}
```

Uso del período de vida (TTL) en DynamoDB

Tiempo de vida (TTL) para DynamoDB es un método rentable para eliminar elementos que ya no son relevantes. TTL permite definir una marca de tiempo de vencimiento por elemento que indica cuándo deja de necesitarse un elemento. DynamoDB elimina automáticamente los elementos vencidos a los pocos días de su fecha de vencimiento, sin afectar al rendimiento de escritura.

Para usar TTL, primero debe activarlo en una tabla y, a continuación, definir un atributo específico para almacenar la marca de tiempo de vencimiento de TTL. La marca de tiempo debe almacenarse en el formato de tiempo Epoch de Unix con nivel de precisión de segundos. Cada vez que se crea o actualiza un elemento, puede calcular el tiempo de vencimiento y guardarlo en el atributo TTL.

El sistema puede eliminar los elementos con atributos TTL válidos y vencidos en cualquier momento, normalmente a los pocos días de que hayan caducado. Puede seguir actualizando los elementos

vencidos que no se hayan eliminado todavía, lo que incluye cambiar o eliminar los atributos TTL. Al actualizar un elemento vencido, le recomendamos que utilice una expresión de condición para asegurarse de que el elemento no se haya eliminado posteriormente. Utilice expresiones de filtro para eliminar los elementos vencidos de los resultados de Scan y Query.

Los elementos eliminados funcionan de forma similar a los que se eliminan con las operaciones de eliminación habituales. Una vez eliminados, los elementos pasan a DynamoDB Streams como eliminaciones de servicio en lugar de eliminaciones de usuario. Además, se eliminan de los índices secundarios locales y de los índices secundarios globales de igual modo que con las otras operaciones de eliminación.

Si usa la versión 2019.11.21 (actual) de las tablas globales y también la característica TTL, DynamoDB replicará las eliminaciones de TTL en todas las tablas de réplica. La eliminación de TTL inicial no consume unidades de capacidad de escritura (WCU) en la región donde se produce el vencimiento de TTL. Sin embargo, la eliminación de TTL replicada en las tablas de réplica consume una unidad de capacidad de escritura replicada cuando se usa la capacidad aprovisionada, o una unidad de escritura replicada cuando se usa el modo de capacidad bajo demanda, en cada una de las regiones de réplica. En estos casos, se aplicarán los cargos pertinentes.

Para obtener más información acerca de TTL, consulte estos temas:

Temas

- Habilitación del período de vida (TTL) de DynamoDB
- Período de vida (TTL) de computación en DynamoDB
- Uso de elementos caducados y el período de vida (TTL)

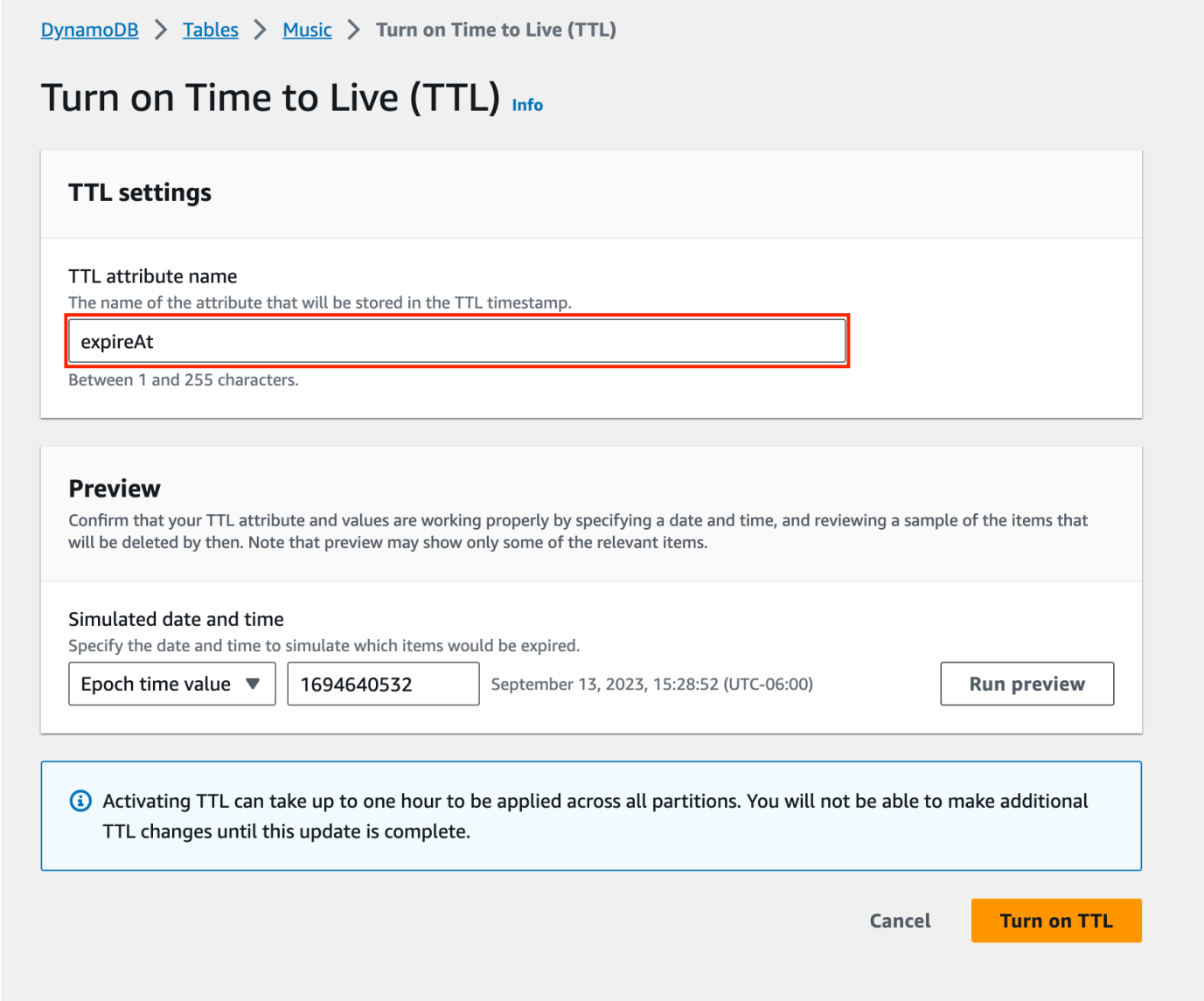
Habilitación del período de vida (TTL) de DynamoDB

> **Note**
>
> Para facilitar la depuración y la verificación del correcto funcionamiento de la característica TTL, los valores proporcionados para el TTL de un elemento se registran en texto no cifrado en los registros de diagnóstico de DynamoDB.

Puede habilitar TTL en la consola de Amazon DynamoDB, la AWS Command Line Interface (AWS CLI) o mediante la referencia de la API de Amazon DynamoDB con cualquiera de los AWS SDK supuestos. La habilitación de TTL en todas las particiones tarda aproximadamente una hora.

Activación del TTL de DynamoDB mediante la consola de AWS

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. Elija Tables (Tablas) y, a continuación, seleccione la tabla que desee modificar.
3. En la pestaña Configuración adicional, dentro de la sección Tiempo de vida (TTL), elija Activar.
4. Al habilitar TTL en una tabla, DynamoDB requiere que identifique un nombre de atributo específico que el servicio buscará al determinar si un elemento cumple los requisitos para el vencimiento. El nombre del atributo TTL, que se muestra a continuación, distingue entre mayúsculas y minúsculas y debe coincidir con el atributo definido en las operaciones de lectura y escritura. Si no coinciden, los elementos caducados no se eliminarán. Si se cambia el nombre del atributo TTL, deberá desactivarse el TTL y volverse a activar con el nuevo atributo de ahora en adelante. Una vez desactivado, el TTL seguirá procesando las eliminaciones durante aproximadamente 30 minutos. El TTL debe reconfigurarse en las tablas restauradas.



5. (Opcional) Puede realizar una prueba simulando la fecha y la hora de vencimiento y haciendo coincidir algunos elementos. Al hacer se muestra una lista de elementos de ejemplo donde se confirma que hay elementos que contienen el nombre del atributo TTL junto con la fecha de vencimiento.

Ahora que TTL está activado, el atributo TTL llevará la marca TTL cuando consulte los elementos en la consola de DynamoDB. Para ver la fecha y la hora de vencimiento de un elemento, mantenga el cursor del ratón sobre el atributo.

Activación del TTL de DynamoDB mediante la API

Python

Puede activar el TTL con código con la operación UpdateTimeToLive.

```
import boto3


def enable_ttl(table_name, ttl_attribute_name):
    """
    Enables TTL on DynamoDB table for a given attribute name
        on success, returns a status code of 200
        on error, throws an exception

    :param table_name: Name of the DynamoDB table
    :param ttl_attribute_name: The name of the TTL attribute being provided to the
 table.
    """
    try:
        dynamodb = boto3.client('dynamodb')

        # Enable TTL on an existing DynamoDB table
        response = dynamodb.update_time_to_live(
            TableName=table_name,
            TimeToLiveSpecification={
                'Enabled': True,
                'AttributeName': ttl_attribute_name
            }
        )

        # In the returned response, check for a successful status code.
        if response['ResponseMetadata']['HTTPStatusCode'] == 200:
            print("TTL has been enabled successfully.")
        else:
            print(f"Failed to enable TTL, status code {response['ResponseMetadata']
['HTTPStatusCode']}")
    except Exception as ex:
        print("Couldn't enable TTL in table %s. Here's why: %s" % (table_name, ex))
        raise


# your values
```

```
enable_ttl('your-table-name', 'expirationDate')
```

Puede confirmar que el TTL está activado mediante la operación DescribeTimeToLive, que describe el estado del TTL en una tabla. El estado de `TimeToLive` es `ENABLED` o `DISABLED`.

```
# create a DynamoDB client
dynamodb = boto3.client('dynamodb')

# set the table name
table_name = 'YourTable'

# describe TTL
response = dynamodb.describe_time_to_live(TableName=table_name)
```

JavaScript

Puede activar el TTL con código con la operación UpdateTimeToLiveCommand.

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-dynamodb";

const enableTTL = async (tableName, ttlAttribute) => {

    const client = new DynamoDBClient({});

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: true,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL enabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to enable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
```

```
        console.error(`Error enabling TTL: ${e}`);
        throw e;
    }
};

// call with your own values
enableTTL('ExampleTable', 'exampleTtlAttribute');
```

Activación del tiempo de vida mediante la AWS CLI

1. Habilite TTL en la tabla `TTLExample`.

   ```
   aws dynamodb update-time-to-live --table-name TTLExample --time-to-live-
   specification "Enabled=true, AttributeName=ttl"
   ```

2. Describa TTL en la tabla `TTLExample`.

   ```
   aws dynamodb describe-time-to-live --table-name TTLExample
   {
       "TimeToLiveDescription": {
           "AttributeName": "ttl",
           "TimeToLiveStatus": "ENABLED"
       }
   }
   ```

3. Añada un elemento a la tabla `TTLExample` con el atributo de período de vida establecido utilizando el shell de BASH y la AWS CLI.

   ```
   EXP=`date -d '+5 days' +%s`
   aws dynamodb put-item --table-name "TTLExample" --item '{"id": {"N": "1"}, "ttl":
    {"N": "'$EXP'"}}'
   ```

Este ejemplo comienza con la fecha actual y añade cinco días para crear una fecha de vencimiento. A continuación, convierte la fecha de vencimiento al formato de tiempo Unix y, por último, agrega un elemento a la tabla "`TTLExample`".

> **Note**
>
> Una forma de establecer valores de vencimiento para período de vida consiste en calcular el número de segundos que se sumarán al momento del vencimiento. Por ejemplo, 5 días son

432 000 segundos. Sin embargo, a menudo es preferible comenzar por una fecha y tomarla como punto de partida.

Es bastante sencillo obtener el tiempo actual en formato de tiempo Unix, como en los siguientes ejemplos.

- Terminal Linux: `date +%s`
- Python: `import time; int(time.time())`
- Java: `System.currentTimeMillis() / 1000L`
- JavaScript: `Math.floor(Date.now() / 1000)`

Activación del TTL de DynamoDB mediante CloudFormation

```
AWSTemplateFormatVersion: "2010-09-09"
Resources:
  TTLExampleTable:
    Type: AWS::DynamoDB::Table
    Description: "A DynamoDB table with TTL Specification enabled"
    Properties:
      AttributeDefinitions:
        - AttributeName: "Album"
          AttributeType: "S"
        - AttributeName: "Artist"
          AttributeType: "S"
      KeySchema:
        - AttributeName: "Album"
          KeyType: "HASH"
        - AttributeName: "Artist"
          KeyType: "RANGE"
      ProvisionedThroughput:
        ReadCapacityUnits: "5"
        WriteCapacityUnits: "5"
      TimeToLiveSpecification:
        AttributeName: "TTLExampleAttribute"
        Enabled: true
```

Puede encontrar aquí más información sobre el uso del TTL en las plantillas de CloudFormation.

Período de vida (TTL) de computación en DynamoDB

Una forma habitual de implementar el TTL consiste en establecer una fecha de vencimiento para los elementos en función de cuándo se crearon o se actualizaron por última vez. Esto se puede hacer añadiendo la hora a las marcas de tiempo `createdAt` y `updatedAt`. Por ejemplo, el TTL de los elementos que se acaban de crear se puede establecer en `createdAt` más 90 días. Cuando se actualiza el elemento, el TTL se puede volver a calcular como `updatedAt` más de 90 días.

El tiempo de vencimiento calculado debe estar en formato Epoch, expresado en segundos. Para que se tenga en cuenta el vencimiento y la eliminación, el TTL no puede tener más de cinco años de antigüedad. Si utiliza cualquier otro formato, los procesos de TTL ignoran el elemento. Si establece la fecha de vencimiento en algún momento del futuro en que quiere que caduque el elemento, caducará cuando llegue dicha fecha. Por ejemplo, supongamos que establece la fecha de vencimiento para el 1724241326 (que es el lunes 21 de agosto de 2024 a las 11:55:26 [GMT]). El elemento vencerá cuando se alcance la fecha especificada.

Temas

- Creación de un elemento y determinación del tiempo de vida
- Actualización de un elemento y del tiempo de vida

Creación de un elemento y determinación del tiempo de vida

En el siguiente ejemplo se muestra cómo calcular el tiempo de vencimiento al crear un elemento nuevo, utilizando `expireAt` como nombre de atributo TTL. Una instrucción de asignación obtiene la hora actual como variable. En el ejemplo, la fecha de vencimiento se calcula en 90 días a partir de la fecha actual. A continuación, la fecha se convierte al formato Epoch y se guarda como un tipo de dato entero en el atributo TTL.

Los siguientes ejemplos de código muestran cómo crear un elemento con TTL.

Java

SDK para Java 2.x

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

/**
 * Creates an item in a DynamoDB table with TTL attributes.
 * This class demonstrates how to add TTL expiration timestamps to DynamoDB
 items.
 */
public class CreateTTL {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
                sortKey - The name of the sort key. Also known as the range
 attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
 is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
    private static final String CREATION_DATE_ATTR = "creationDate";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String SUCCESS_MESSAGE = "%s PutItem operation with TTL
 successful.";
    private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
 DynamoDB table \"%s\" can't be found.";

    private final DynamoDbClient dynamoDbClient;
```

```
    /**
     * Constructs a CreateTTL instance with the specified DynamoDB client.
     *
     * @param dynamoDbClient The DynamoDB client to use
     */
    public CreateTTL(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }

    /**
     * Constructs a CreateTTL with a default DynamoDB client.
     */
    public CreateTTL() {
        this.dynamoDbClient = null;
    }

    /**
     * Main method to demonstrate creating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new CreateTTL().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and create an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);
```

```
        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final CreateTTL createTTL = new CreateTTL(ddb);
            createTTL.createItemWithTTL(tableName, primaryKey, sortKey);
            return 0;
        } catch (Exception e) {
            throw e;
        }
    }

    /**
     * Creates an item in the specified table with TTL attributes.
     *
     * @param tableName The name of the table
     * @param primaryKeyValue The value for the primary key
     * @param sortKeyValue The value for the sort key
     * @return The response from the PutItem operation
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     */
    public PutItemResponse createItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
sortKeyValue) {
        // Get current time in epoch second format
        final long createDate = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = createDate + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        final Map<String, AttributeValue> itemMap = new HashMap<>();
        itemMap.put(
            PRIMARY_KEY_ATTR,
AttributeValue.builder().s(primaryKeyValue).build());
```

```
        itemMap.put(SORT_KEY_ATTR,
 AttributeValue.builder().s(sortKeyValue).build());
        itemMap.put(
            CREATION_DATE_ATTR,
            AttributeValue.builder().n(String.valueOf(createDate)).build());
        itemMap.put(
            EXPIRE_AT_ATTR,
            AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final PutItemRequest request =
            PutItemRequest.builder().tableName(tableName).item(itemMap).build();

        try {
            final PutItemResponse response = dynamoDbClient.putItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

```
import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";

export function createDynamoDBItem(table_name, region, partition_key, sort_key) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });
```

```
    // Get the current time in epoch second format
    const current_time = Math.floor(new Date().getTime() / 1000);

    // Calculate the expireAt time (90 days from now) in epoch second format
    const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 *
 1000) / 1000);

    // Create DynamoDB item
    const item = {
        'partitionKey': {'S': partition_key},
        'sortKey': {'S': sort_key},
        'createdAt': {'N': current_time.toString()},
        'expireAt': {'N': expire_at.toString()}
    };

    const putItemCommand = new PutItemCommand({
        TableName: table_name,
        Item: item,
        ProvisionedThroughput: {
            ReadCapacityUnits: 1,
            WriteCapacityUnits: 1,
        },
    });

    client.send(putItemCommand, function(err, data) {
        if (err) {
            console.log("Exception encountered when creating item %s, here's what
 happened: ", data, err);
            throw err;
        } else {
            console.log("Item created successfully: %s.", data);
            return data;
        }
    });
}

// Example usage (commented out for testing)
// createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

```
from datetime import datetime, timedelta

import boto3


def create_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Creates a DynamoDB item with an attached expiry attribute.

    :param table_name: Table name for the boto3 resource to target when creating
 an item
    :param region: string representing the AWS region. Example: `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expiration time (90 days from now) in epoch second format
        expiration_time = int((datetime.now() + timedelta(days=90)).timestamp())

        item = {
            "primaryKey": primary_key,
            "sortKey": sort_key,
            "creationDate": current_time,
            "expireAt": expiration_time,
        }
        response = table.put_item(Item=item)

        print("Item created successfully.")
        return response
    except Exception as e:
        print(f"Error creating item: {e}")
        raise e
```

```
# Use your own values
create_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
)
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para Python (Boto3).

Actualización de un elemento y del tiempo de vida

Este ejemplo es la continuación del que aparece en la sección anterior. La fecha de vencimiento se puede volver a calcular si se actualiza el elemento. En el siguiente ejemplo, se vuelve a calcular la marca de tiempo `expireAt` para que sea de 90 días a partir de la fecha actual.

En los siguientes ejemplos de código, se muestra cómo actualizar el TTL de un elemento.

Java

SDK para Java 2.x

Actualice el TTL en un elemento de DynamoDB existente en una tabla.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

    public UpdateItemResponse updateItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
 sortKeyValue) {
        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;
```

```
        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = new HashMap<>();
        keyMap.put(PRIMARY_KEY_ATTR,
AttributeValue.builder().s(primaryKeyValue).build());
        keyMap.put(SORT_KEY_ATTR,
AttributeValue.builder().s(sortKeyValue).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            ":c",
AttributeValue.builder().n(String.valueOf(currentTime)).build());
        expressionAttributeValues.put(
            ":e",
AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final UpdateItemResponse response =
dynamoDbClient.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItem = async (tableName, partitionKey, sortKey, region = 'us-
east-1') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);
    const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            partitionKey: partitionKey,
            sortKey: sortKey
        }),
        UpdateExpression: "SET updatedAt = :c, expireAt = :e",
        ExpressionAttributeValues: marshall({
            ":c": currentTime,
            ":e": expireAt
        }),
    };

    try {
        const data = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(data.Attributes);
        console.log("Item updated successfully: %s", responseData);
        return responseData;
    } catch (err) {
        console.error("Error updating item:", err);
        throw err;
    }
}
```

```
// Example usage (commented out for testing)
// updateItem('your-table-name', 'your-partition-key-value', 'your-sort-key-
value');
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

```
from datetime import datetime, timedelta

import boto3


def update_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Update an existing DynamoDB item with a TTL.
    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        # Create the DynamoDB resource.
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expireAt time (90 days from now) in epoch second format
        expire_at = int((datetime.now() + timedelta(days=90)).timestamp())

        table.update_item(
            Key={"partitionKey": primary_key, "sortKey": sort_key},
            UpdateExpression="set updatedAt=:c, expireAt=:e",
```

```
            ExpressionAttributeValues={":c": current_time, ":e": expire_at},
        )

        print("Item updated successfully.")
    except Exception as e:
        print(f"Error updating item: {e}")


# Replace with your own values
update_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
)
```

- Para obtener información la API, consulte UpdateItem en la referencia de la API de AWS SDK para Python (Boto3).

Los ejemplos de TTL que se analizan en esta introducción muestran un método para garantizar que en la tabla solo se guarden los elementos actualizados recientemente. La vida útil de los elementos actualizados se prolonga, mientras que los que no se actualizan tras su creación caducan y se eliminan sin costo alguno, lo que reduce el almacenamiento y mantiene las tablas limpias.

Uso de elementos caducados y el período de vida (TTL)

Los elementos vencidos que aún deben eliminarse se pueden filtrar de las operaciones de lectura y escritura. Esto resulta útil en situaciones en las que los datos vencidos ya no son válidos y no deben utilizarse. Si no están filtrados, seguirán mostrándose en las operaciones de lectura y escritura hasta que se eliminen en el proceso en segundo plano.

> **Note**
>
> Estos elementos seguirán contabilizándose para los costos de almacenamiento y lectura hasta que se eliminen.

Las eliminaciones de TTL se pueden identificar en DynamoDB Streams, pero solo en la región en la que se han eliminado. Las eliminaciones de TTL que se replican en las regiones de la tabla global no se pueden identificar en las transmisiones de DynamoDB de las regiones en las que se replica la eliminación.

Filtrado de los elementos vencidos de las operaciones de lectura

Para las operaciones de lectura, como Scan y Query, una expresión de filtro puede filtrar los elementos vencidos que aún no se han eliminado. Como se muestra en el siguiente fragmento de código, la expresión de filtro puede filtrar los elementos en los que el tiempo de TTL sea igual o inferior al tiempo actual. Por ejemplo, el código del SDK de Python incluye una declaración de asignación que obtiene la hora actual como una variable (now) y la convierte en `int` para el formato de tiempo Epoch.

En los siguientes ejemplos de código, se muestra cómo consultar elementos de TTL.

Java

SDK para Java 2.x

Consulte una expresión filtrada para recopilar elementos de TTL en una tabla de DynamoDB mediante AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.Map;
import java.util.Optional;

        final QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final QueryResponse response = ddb.query(request);
            System.out.println("Query successful. Found " + response.count() + "
 items that have not expired yet.");
```

```
            // Print each item
            response.items().forEach(item -> {
                System.out.println("Item: " + item);
            });

            return 0;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte una expresión filtrada para recopilar elementos de TTL en una tabla de DynamoDB mediante AWS SDK para JavaScript.

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const queryFiltered = async (tableName, primaryKey, region = 'us-east-1')
 => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pk",
        FilterExpression: "#ea > :ea",
```

```
        ExpressionAttributeNames: {
            "#pk": "primaryKey",
            "#ea": "expireAt"
        },
        ExpressionAttributeValues: marshall({
            ":pk": primaryKey,
            ":ea": currentTime
        })
    };

    try {
        const { Items } = await client.send(new QueryCommand(params));
        Items.forEach(item => {
            console.log(unmarshall(item))
        });
        return Items;
    } catch (err) {
        console.error(`Error querying items: ${err}`);
        throw err;
    }
}

// Example usage (commented out for testing)
// queryFiltered('your-table-name', 'your-partition-key-value');
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte una expresión filtrada para recopilar elementos de TTL en una tabla de DynamoDB mediante AWS SDK for Python (Boto3).

```
from datetime import datetime

import boto3


def query_dynamodb_items(table_name, partition_key):
```

```
    """

    :param table_name: Name of the DynamoDB table
    :param partition_key:
    :return:
    """
    try:
        # Initialize a DynamoDB resource
        dynamodb = boto3.resource("dynamodb", region_name="us-east-1")

        # Specify your table
        table = dynamodb.Table(table_name)

        # Get the current time in epoch format
        current_time = int(datetime.now().timestamp())

        # Perform the query operation with a filter expression to exclude expired
 items
        # response = table.query(
        #
 KeyConditionExpression=boto3.dynamodb.conditions.Key('partitionKey').eq(partition_key),
        #
 FilterExpression=boto3.dynamodb.conditions.Attr('expireAt').gt(current_time)
        # )
        response = table.query(

 KeyConditionExpression=dynamodb.conditions.Key("partitionKey").eq(partition_key),

 FilterExpression=dynamodb.conditions.Attr("expireAt").gt(current_time),
        )

        # Print the items that are not expired
        for item in response["Items"]:
            print(item)

    except Exception as e:
        print(f"Error querying items: {e}")


# Call the function with your values
query_dynamodb_items("Music", "your-partition-key-value")
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Python (Boto3).

Escritura condicional en los elementos vencidos

Se puede usar una expresión condicional para evitar escrituras en los elementos vencidos. El siguiente fragmento de código es una actualización condicional que comprueba si el tiempo de vencimiento es superior al tiempo actual. Si es verdadero, la operación de escritura continuará.

En los siguientes ejemplos de código se muestra cómo actualizar de forma condicional el TTL de un elemento.

Java

SDK para Java 2.x

Actualice el TTL en un elemento de DynamoDB existente en una tabla con una condición.

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import
 software.amazon.awssdk.services.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.Map;
import java.util.Optional;

/**
 * Updates an item in a DynamoDB table with TTL attributes using a conditional
 expression.
 * This class demonstrates how to conditionally update TTL expiration timestamps.
 */
public class UpdateTTLConditional {
```

```
   private static final String USAGE =
       """
           Usage:
               <tableName> <primaryKey> <sortKey> <region>
           Where:
               tableName - The Amazon DynamoDB table being queried.
               primaryKey - The name of the primary key. Also known as the hash
or partition key.
               sortKey - The name of the sort key. Also known as the range
attribute.
               region (optional) - The AWS region that the Amazon DynamoDB table
is located in. (Default: us-east-1)
           """;
   private static final int DAYS_TO_EXPIRE = 90;
   private static final int SECONDS_PER_DAY = 24 * 60 * 60;
   private static final String PRIMARY_KEY_ATTR = "primaryKey";
   private static final String SORT_KEY_ATTR = "sortKey";
   private static final String UPDATED_AT_ATTR = "updatedAt";
   private static final String EXPIRE_AT_ATTR = "expireAt";
   private static final String UPDATE_EXPRESSION = "SET " + UPDATED_AT_ATTR +
"=:c, " + EXPIRE_AT_ATTR + "=:e";
   private static final String CONDITION_EXPRESSION = "attribute_exists(" +
PRIMARY_KEY_ATTR + ")";
   private static final String SUCCESS_MESSAGE = "%s UpdateItem operation with
TTL successful.";
   private static final String CONDITION_FAILED_MESSAGE = "Condition check
failed. Item does not exist.";
   private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
DynamoDB table \"%s\" can't be found.";

   private final DynamoDbClient dynamoDbClient;

   /**
    * Constructs an UpdateTTLConditional with a default DynamoDB client.
    */
   public UpdateTTLConditional() {
       this.dynamoDbClient = null;
   }

   /**
    * Constructs an UpdateTTLConditional with the specified DynamoDB client.
    *
    * @param dynamoDbClient The DynamoDB client to use
    */
```

```
    public UpdateTTLConditional(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }

    /**
     * Main method to demonstrate conditionally updating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new UpdateTTLConditional().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and conditionally update an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);

        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;
```

```
        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = Map.of(
            PRIMARY_KEY_ATTR, AttributeValue.builder().s(primaryKey).build(),
            SORT_KEY_ATTR, AttributeValue.builder().s(sortKey).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = Map.of(
            ":c",
 AttributeValue.builder().n(String.valueOf(currentTime)).build(),
            ":e",
 AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .conditionExpression(CONDITION_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final UpdateItemResponse response = ddb.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return 0;
        } catch (ConditionalCheckFailedException e) {
            System.err.println(CONDITION_FAILED_MESSAGE);
            throw e;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Actualice el TTL en un elemento de DynamoDB existente en una tabla con una condición.

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItemConditional = async (tableName, partitionKey, sortKey,
 region = 'us-east-1', newAttribute = 'default-value') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            artist: partitionKey,
            album: sortKey
        }),
        UpdateExpression: "SET newAttribute = :newAttribute",
        ConditionExpression: "expireAt > :expiration",
        ExpressionAttributeValues: marshall({
            ':newAttribute': newAttribute,
            ':expiration': currentTime
        }),
        ReturnValues: "ALL_NEW"
    };

    try {
        const response = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(response.Attributes);
        console.log("Item updated successfully: ", responseData);
        return responseData;
    } catch (error) {
        if (error.name === "ConditionalCheckFailedException") {
```

```
            console.log("Condition check failed: Item's 'expireAt' is expired.");
        } else {
            console.error("Error updating item: ", error);
        }
        throw error;
    }
};

// Example usage (commented out for testing)
// updateItemConditional('your-table-name', 'your-partition-key-value', 'your-
sort-key-value');
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Actualice el TTL en un elemento de DynamoDB existente en una tabla con una condición.

```
from datetime import datetime, timedelta

import boto3
from botocore.exceptions import ClientError


def update_dynamodb_item_ttl(table_name, region, primary_key, sort_key,
 ttl_attribute):
    """
    Updates an existing record in a DynamoDB table with a new or updated TTL
 attribute.

    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :param ttl_attribute: name of the TTL attribute in the target DynamoDB table
    :return:
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
```

```
        table = dynamodb.Table(table_name)

        # Generate updated TTL in epoch second format
        updated_expiration_time = int((datetime.now() +
 timedelta(days=90)).timestamp())

        # Define the update expression for adding/updating a new attribute
        update_expression = "SET newAttribute = :val1"

        # Define the condition expression for checking if 'expireAt' is not
 expired
        condition_expression = "expireAt > :val2"

        # Define the expression attribute values
        expression_attribute_values = {":val1": ttl_attribute, ":val2":
 updated_expiration_time}

        response = table.update_item(
            Key={"primaryKey": primary_key, "sortKey": sort_key},
            UpdateExpression=update_expression,
            ConditionExpression=condition_expression,
            ExpressionAttributeValues=expression_attribute_values,
        )

        print("Item updated successfully.")
        return response["ResponseMetadata"]["HTTPStatusCode"]  # Ideally a 200 OK
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            print("Condition check failed: Item's 'expireAt' is expired.")
        else:
            print(f"Error updating item: {e}")
    except Exception as e:
        print(f"Error updating item: {e}")


# replace with your values
update_dynamodb_item_ttl(
    "your-table-name",
    "us-east-1",
    "your-partition-key-value",
    "your-sort-key-value",
    "your-ttl-attribute-value",
)
```

- Para obtener información la API, consulte UpdateItem en la referencia de la API de AWS SDK para Python (Boto3).

Identificación de elementos eliminados en DynamoDB Streams

El registro de secuencias contiene el campo de identidad del usuario `Records[<index>].userIdentity`. Los elementos que elimina el proceso TTL tienen los campos siguientes:

```
Records[<index>].userIdentity.type
"Service"

Records[<index>].userIdentity.principalId
"dynamodb.amazonaws.com"
```

En el código JSON siguiente se muestra la parte pertinente de un registro de secuencias único:

```
"Records": [
  {
 ...
  "userIdentity": {
  "type": "Service",
       "principalId": "dynamodb.amazonaws.com"
    }
   ...
 }
]
```

Consulta de tablas en DynamoDB

Puede usar la operación de la API `Query` en Amazon DynamoDB para buscar elementos según los valores de clave principal.

Debe proporcionar el nombre del atributo de clave de partición y un único valor para dicho atributo. `Query` devuelve todos los elementos que contengan ese valor de clave de partición. Si lo desea, puede proporcionar un atributo de clave de ordenación y utilizar un operador de comparación para limitar los resultados de búsqueda.

Para obtener más información sobre cómo usar `Query`, como la sintaxis de la solicitud, los parámetros de respuesta y ejemplos adicionales, consulte Query en la Referencia de la API de Amazon DynamoDB.

Temas

- Expresiones de condición de clave para la operación Query en DynamoDB
- Expresiones de filtro para la operación Query en DynamoDB
- Paginación de los resultados de consulta de tabla en DynamoDB
- Otros aspectos del trabajo con la operación Query en DynamoDB

Expresiones de condición de clave para la operación Query en DynamoDB

Puede utilizar cualquier nombre de atributo en una expresión de condición de clave, siempre y cuando el primer carácter sea `a-z` o `A-Z` y el resto de los caracteres (a partir del segundo carácter, si lo hay) sea `a-z`, `A-Z` o `0-9`. Además, el nombre de atributo no debe ser una palabra reservada de DynamoDB. Para obtener una lista completa de estas palabras, consulte Palabras reservadas en DynamoDB. Si un nombre de atributo no cumple estos requisitos, debe definir un nombre de atributo de expresión como marcador de posición. Para obtener más información, consulte Nombres de atributos de expresión (Alias) en DynamoDB.

DynamoDB almacena cerca unos de otros y ordenados según su clave de ordenación aquellos elementos que tienen un valor de clave de partición determinado. En una operación `Query`, DynamoDB recupera los elementos de forma ordenada y, a continuación, los procesa mediante las expresiones `KeyConditionExpression` y `FilterExpression` que estén presentes. Solo entonces devuelve los resultados de `Query` al cliente.

Una operación `Query` siempre devuelve un conjunto de resultados. Si no se encuentran elementos coincidentes, el conjunto de resultados está vacío.

Los resultados de `Query` siempre se ordenan según el valor de la clave de ordenación. Si el tipo de datos de la clave de ordenación es `Number`, los resultados se devuelven en orden numérico. De lo contrario, los resultados se devuelven según el orden de los bytes UTF-8. De forma predeterminada, el orden es ascendente. Para invertirlo, establezca el parámetro `ScanIndexForward` en `false`.

En una sola operación `Query` se puede recuperar un máximo de 1 MB de datos. Este límite se aplica antes de que `FilterExpression` o `ProjectionExpression` se apliquen a los resultados. Si `LastEvaluatedKey` está presente en la respuesta y su valor no es null, debe paginar el conjunto de resultados (consulte Paginación de los resultados de consulta de tabla en DynamoDB).

Ejemplos de expresión de condición de clave

Para especificar los criterios de búsqueda, se utiliza una expresión de condición de clave; se trata de una cadena que determina los elementos que se van a leer en la tabla o el índice.

Debe especificar el nombre y valor de la clave de partición como una condición de igualdad. No puede utilizar un atributo no clave en una expresión de condición clave.

Si lo desea, puede proporcionar una segunda condición para la clave de ordenación (en caso de incluirse). En la condición de clave de ordenación se debe utilizar uno de los siguientes operadores de comparación:

- *a* = *b*: es true si el atributo *a* es igual al valor *b*.
- *a* < *b*: es true si *a* es menor que *b*.
- *a* <= *b*: es true si *a* es menor o igual que *b*.
- *a* > *b*: es true si *a* es mayor que *b*.
- *a* >= *b*: es true si *a* es mayor o igual que *b*.
- *a* `BETWEEN` *b* `AND` *c*: es true si *a* es mayor o igual que *b* y menor o igual que *c*.

También se admite la siguiente función:

- `begins_with (`*a*`, `*substr*`)`: es true si el valor del atributo *a* comienza por una subcadena determinada.

En los siguientes ejemplos de la AWS Command Line Interface (AWS CLI) se muestra el uso de expresiones de condición de clave. Estas expresiones utilizan marcadores de posición (como `:name` y `:sub`) en lugar de los valores reales. Para obtener más información, consulte Nombres de atributos de expresión (Alias) en DynamoDB y Uso de valores de atributos de expresión en DynamoDB.

Example

Consulta la tabla `Thread` para buscar un valor concreto de `ForumName` (clave de partición). La consulta leerá todos los elementos que tengan ese valor de `ForumName`, porque la clave de ordenación (`Subject`) no se incluye en `KeyConditionExpression`.

```
aws dynamodb query \
    --table-name Thread \
```

```
    --key-condition-expression "ForumName = :name" \
    --expression-attribute-values  '{":name":{"S":"Amazon DynamoDB"}}'
```

Example

Consulta la tabla `Thread` para buscar un valor concreto de `ForumName` (clave de partición), pero esta vez devuelve solo los elementos que tienen un valor determinado de `Subject` (clave de ordenación).

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :name and Subject = :sub" \
    --expression-attribute-values  file://values.json
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo `values.json`.

```
{
    ":name":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"}
}
```

Example

Consulta la tabla `Reply` para buscar un valor concreto de `Id` (clave de partición), pero solo devuelve los elementos cuyo valor de `ReplyDateTime` (clave de ordenación) comienza por determinados caracteres.

```
aws dynamodb query \
    --table-name Reply \
    --key-condition-expression "Id = :id and begins_with(ReplyDateTime, :dt)" \
    --expression-attribute-values  file://values.json
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo `values.json`.

```
{
    ":id":{"S":"Amazon DynamoDB#DynamoDB Thread 1"},
    ":dt":{"S":"2015-09"}
```

```
}
```

Expresiones de filtro para la operación Query en DynamoDB

Si tiene que refinar más los resultados de `Query`, si lo desea puede indicar una expresión de filtro. Una expresión de filtro determina qué elementos de los resultados de `Query` se deben devolver al usuario. Todos los demás resultados se descartan.

Una expresión de filtro se aplica después de que la operación `Query` haya finalizado, pero antes de devolver los resultados. Por consiguiente, `Query` consume la misma cantidad de capacidad de lectura aunque se especifique una expresión de filtro.

En una operación `Query` se puede recuperar un máximo de 1 MB de datos. Este límite se aplica antes de evaluar la expresión de filtro.

Una expresión de filtro no puede contener atributos de clave de partición ni de clave de ordenación. Esos atributos se deben especificar en la expresión de condición de clave, no en la expresión de filtro.

La sintaxis de una expresión de filtro es similar a la de una expresión de condición de clave. Las expresiones de filtro pueden utilizar los mismos comparadores, funciones y operadores lógicos que las expresiones de condición de clave. Además, las expresiones de filtro pueden usar los operadores “no es igual” (<>), `OR`, `CONTAINS`, `IN`, `BEGINS_WITH`, `BETWEEN`, `EXISTS` y `SIZE`. Para obtener más información, consulte Expresiones de condición de clave para la operación Query en DynamoDB y Sintaxis de las expresiones de filtro y condición.

Example

En el siguiente ejemplo de la AWS CLI, se consulta la tabla `Thread` para buscar un valor concreto de `ForumName` (clave de partición) y `Subject` (clave de ordenación). De los elementos encontrados, solo se devuelven las conversaciones más populares; es decir, solo aquellas que tienen un valor de `Views` mayor que el número especificado.

```
aws dynamodb query \
    --table-name Thread \
    --key-condition-expression "ForumName = :fn and Subject begins_with :sub" \
    --filter-expression "#v >= :num" \
    --expression-attribute-names '{"#v": "Views"}' \
    --expression-attribute-values file://values.json
```

Los argumentos de `--expression-attribute-values` se almacenan en el archivo `values.json`.

```
{
    ":fn":{"S":"Amazon DynamoDB"},
    ":sub":{"S":"DynamoDB Thread 1"},
    ":num":{"N":"3"}
}
```

Tenga en cuenta que `Views` es una palabra reservada en DynamoDB (consulte Palabras reservadas en DynamoDB), por lo que en este ejemplo se utiliza `#v` como marcador de posición. Para obtener más información, consulte Nombres de atributos de expresión (Alias) en DynamoDB.

> **Note**
>
> Una expresión de filtro elimina los elementos del conjunto de resultados de `Query`. Si es posible, evite usar `Query` cuando prevea que va a recuperar gran cantidad de elementos, pero que tendrá que descartar la mayoría de ellos.

Paginación de los resultados de consulta de tabla en DynamoDB

DynamoDB pagina los resultados de las operaciones `Query`. La paginación permite dividir los resultados de `Query` en "páginas" de datos con un tamaño de 1 MB (o menos). Una aplicación puede procesar la primera página de resultados, luego la segunda página, y así sucesivamente.

Una operación `Query` única devuelve solamente un conjunto de resultados que se ajuste al límite de tamaño de 1 MB. Para determinar si hay más resultados y para recuperarlos de página en página, las aplicaciones deben hacer lo siguiente:

1. Examinar el resultado de `Query` de bajo nivel:
   - Si el resultado contiene una entrada `LastEvaluatedKey`, vaya al paso 2.
   - Si no figura `LastEvaluatedKey` en el resultado, no hay más elementos que recuperar.
2. Construya una `Query` con la misma `KeyConditionExpression`. Sin embargo, esta vez, se toma el valor `LastEvaluatedKey` del paso 1 y se usa como parámetro `ExclusiveStartKey` en la nueva solicitud `Query`.
3. Ejecutar la nueva solicitud `Query`.
4. Ir al paso 1.

Es decir, el valor de `LastEvaluatedKey` de la respuesta de `Query` debe utilizarse como valor de `ExclusiveStartKey` en la siguiente solicitud `Query`. Si no hay una entrada `LastEvaluatedKey` en una respuesta de `Query`, significa que se ha recuperado la última página de resultados. Si `LastEvaluatedKey` no está vacía, no significa necesariamente que haya más datos en el conjunto de resultados. La única forma de saber que se ha alcanzado el final del conjunto de resultados es cuando `LastEvaluatedKey` está vacío.

Puede utilizar la AWS CLI para ver este comportamiento. La AWS CLI envía solicitudes `Query` de bajo nivel a DynamoDB una y otra vez hasta que `LastEvaluatedKey` ya no esté presente en los resultados. Considere el siguiente ejemplo de la AWS CLI que recupera títulos de películas de un determinado año:

```
aws dynamodb query --table-name Movies \
    --projection-expression "title" \
    --key-condition-expression "#y = :yyyy" \
    --expression-attribute-names '{"#y":"year"}' \
    --expression-attribute-values '{":yyyy":{"N":"1993"}}' \
    --page-size 5 \
    --debug
```

Normalmente, la AWS CLI se encarga de la paginación automáticamente. No obstante, en este ejemplo, el parámetro `--page-size` de la AWS CLI limita el número de elementos por página. El parámetro `--debug` muestra información de bajo nivel de las solicitudes y las respuestas.

Si ejecuta el ejemplo, la primera respuesta de DynamoDB será similar a la siguiente.

```
2017-07-07 11:13:15,603 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":5,"Items":[{"title":{"S":"A Bronx Tale"}},
{"title":{"S":"A Perfect World"}},{"title":{"S":"Addams Family Values"}},
{"title":{"S":"Alive"}},{"title":{"S":"Benny & Joon"}}],
"LastEvaluatedKey":{"year":{"N":"1993"},"title":{"S":"Benny & Joon"}},
"ScannedCount":5}'
```

El elemento `LastEvaluatedKey` de la respuesta indica que no se han recuperado todos los elementos. La AWS CLI envía entonces otra solicitud `Query` a DynamoDB. Este patrón de solicitud y respuesta continúa hasta la respuesta final.

```
2017-07-07 11:13:16,291 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":1,"Items":[{"title":{"S":"What\'s Eating Gilbert
 Grape"}}],"ScannedCount":1}'
```

La ausencia de `LastEvaluatedKey` indica que no hay más elementos que recuperar.

> **Note**
>
> Los SDK de AWS se encargan también de las respuestas de DynamoDB de bajo nivel (incluida la presencia o ausencia de `LastEvaluatedKey`) y proporcionan varias abstracciones para paginar los resultados de `Query`. Por ejemplo, la interfaz de documentos de SDK para Java proporciona compatibilidad con `java.util.Iterator` para que pueda examinar los resultados de uno en uno.
> Para ver ejemplos de código en diversos lenguajes de programación, consulte la Guía de inicio de Amazon DynamoDB y la documentación del SDK de AWS correspondiente a su lenguaje.

También puede reducir el tamaño de página limitando el número de elementos del conjunto de resultados, con el parámetro de `Limit` de la operación de `Query`.

Para obtener más información sobre cómo realizar consultas en DynamoDB, consulte Consulta de tablas en DynamoDB.

Otros aspectos del trabajo con la operación Query en DynamoDB

En esta sección se tratan aspectos adicionales de la operación Query de DynamoDB, como la limitación del tamaño del resultado, el recuento de elementos analizados frente a los devueltos, la supervisión del consumo de capacidad de lectura y el control de la coherencia de lectura.

Limitación del número de elementos del conjunto de resultados

Con la operación `Query`, puede limitar el número de elementos que lee. Para ello, establezca el parámetro `Limit` en el número máximo de elementos que desee.

Por ejemplo, suponga que utiliza `Query` en una tabla con un valor de `Limit` de 6 y sin expresión de filtro. El resultado de `Query` contendrá los seis primeros elementos de la tabla que coincidan con la expresión de condición de clave de la solicitud.

Ahora, suponga que agrega una expresión de filtro a `Query`. En este caso, DynamoDB lee hasta seis elementos y, a continuación, devuelve solo aquellos que coinciden con la expresión de filtro. El resultado final de `Query` contiene seis elementos o menos, aunque más elementos hubieran coincidido con la expresión de filtro si DynamoDB hubiera seguido leyendo más elementos.

Recuento de los elementos en los resultados

Además de los elementos que coinciden con los criterios, la respuesta de `Query` contiene las entradas siguientes:

- `ScannedCount`: número de elementos que coinciden con la expresión de condición de clave antes de aplicar una expresión de filtro (si la hay).
- `Count`: número de elementos restantes después de aplicar una expresión de filtro (si la hay).

> **Note**
>
> Si no se utiliza una expresión de filtro, `ScannedCount` y `Count` tendrán el mismo valor.

Si el tamaño del conjunto de resultados de `Query` es mayor que 1 MB, entonces `ScannedCount` y `Count` representarán únicamente un recuento parcial de los elementos totales. Deberá llevar a cabo varias operaciones `Query` para recuperar todos los resultados (consulte Paginación de los resultados de consulta de tabla en DynamoDB).

Cada respuesta de `Query` contendrá los valores de `ScannedCount` y `Count` de los elementos que se han procesado en esa solicitud `Query` concreta. Para obtener totales globales para todas las solicitudes `Query`, puede llevar un recuento total de `ScannedCount` y `Count`.

Unidades de capacidad consumidas por la consulta

Puede utilizar cualquier tabla o índice secundario `Query`, siempre que proporcione el nombre del atributo de clave de partición y un valor único para ese atributo. `Query` devuelve todos los elementos que contienen ese valor de clave de partición. Opcionalmente, puede proporcionar un atributo de clave de clasificación y utilizar un operador de comparación para limitar los resultados de la búsqueda. `Query` Las operaciones de la API consumen unidades de capacidad de lectura, como se indica a continuación.

Si **`Query`** se aplica a...	DynamoDB consume unidades de capacidad de lectura de...
Tabla	La capacidad de lectura aprovisionada de la tabla.

Si **`Query`** se aplica a...	DynamoDB consume unidades de capacidad de lectura de...
Índice secundario global	La capacidad de lectura aprovisionada del índice.
Índice secundario local	La capacidad de lectura aprovisionada de la tabla base.

De forma predeterminada, una operación `Query` no devuelve datos sobre la cantidad de capacidad de lectura que consume. Sin embargo, puede especificar el parámetro `ReturnConsumedCapacity` en una solicitud `Query` para obtener esta información. A continuación se muestran los ajustes válidos de `ReturnConsumedCapacity`:

- `NONE`: no se devuelven datos de capacidad consumida. (Esta es la opción predeterminada.)
- `TOTAL`: la respuesta incluye el número total de unidades de capacidad de lectura consumidas.
- `INDEXES`: la respuesta muestra el número total de unidades de capacidad de lectura consumidas, así como la capacidad consumida de cada tabla e índice a los que se ha obtenido acceso.

DynamoDB calcula el número de unidades de capacidad de lectura consumidas según el número de elementos y el tamaño de dichos elementos, no según la cantidad de datos que se devuelven a una aplicación. Por este motivo, el número de unidades de capacidad consumidas es el mismo si se solicitan todos los atributos (el comportamiento predeterminado) o solo algunos de ellos (mediante una expresión de proyección). El número también es el mismo tanto si se utiliza una expresión de filtro como si no. `Query`consume una unidad de capacidad de lectura mínima para realizar una lectura altamente coherente por segundo o dos lecturas coherentes posteriores por segundo para un elemento de hasta 4 KB. Para leer un elemento mayor que 4 KB, DynamoDB necesita unidades de solicitud de lectura adicionales. Con las tablas vacías y las tablas muy grandes con una cantidad escasa de claves de partición, es posible que se cobren algunas RCU adicionales además de la cantidad de datos consultados. Esto cubre el costo de atender la solicitud `Query`, incluso si no hay datos.

Coherencia de lectura para la consulta

De forma predeterminada, una operación `Query` lleva a cabo lecturas consistentes finales. Esto significa que los resultados de `Query` podrían no reflejar los cambios provocados por operaciones

`PutItem` o `UpdateItem` realizadas recientemente. Para obtener más información, consulte Coherencia de lectura de DynamoDB.

Si requiere lecturas de consistencia alta, establezca el parámetro `ConsistentRead` en `true` en la solicitud `Query`.

Análisis de tablas en DynamoDB

Una operación `Scan` en Amazon DynamoDB lee todos los elementos de una tabla o de un índice secundario. De manera predeterminada, una operación `Scan` devuelve todos los atributos de datos de todos los elementos de la tabla o el índice. Puede utilizar el parámetro `ProjectionExpression` para que `Scan` solo devuelva algunos de los atributos, en lugar de todos ellos.

`Scan` siempre devuelve un conjunto de resultados. Si no se encuentran elementos coincidentes, el conjunto de resultados está vacío.

En una sola solicitud `Scan` se puede recuperar un máximo de 1 MB de datos. Si lo desea, DynamoDB puede aplicar una expresión de filtro a estos datos, para reducir los resultados antes de que se devuelvan al usuario.

Temas

- Expresiones de filtro para el análisis
- Limitación del número de elementos del conjunto de resultados
- Paginación de los resultados
- Recuento de los elementos en los resultados
- Unidades de capacidad que consume el análisis
- Coherencia de lectura para el análisis
- Análisis paralelo

Expresiones de filtro para el análisis

Si tiene que refinar más los resultados de `Scan`, si lo desea puede indicar una expresión de filtro. Una expresión de filtro determina qué elementos de los resultados de `Scan` se deben devolver al usuario. Todos los demás resultados se descartan.

Una expresión de filtro se aplica después de que la operación `Scan` haya finalizado, pero antes de devolver los resultados. Por consiguiente, `Scan` consume la misma cantidad de capacidad de lectura aunque se especifique una expresión de filtro.

En una operación `Scan` se puede recuperar un máximo de 1 MB de datos. Este límite se aplica antes de evaluar la expresión de filtro.

Con `Scan`, puede especificar cualquier atributo en una expresión de filtro, incluidos los atributos de clave de partición y de clave de ordenación.

La sintaxis de una expresión de filtro es la misma que la de una expresión de condición. Las expresiones de filtro pueden utilizar los comparadores, funciones y operadores lógicos que las expresiones de condición. Consulte Expresiones, operadores y funciones de condición y filtro en DynamoDB para obtener más información sobre operadores lógicos.

Example

En el siguiente ejemplo de la AWS Command Line Interface (AWS CLI) se examina la tabla `Thread` y solo se devuelven los últimos elementos publicados por un usuario determinado.

```
aws dynamodb scan \
     --table-name Thread \
     --filter-expression "LastPostedBy = :name" \
     --expression-attribute-values '{":name":{"S":"User A"}}'
```

Limitación del número de elementos del conjunto de resultados

La operación `Scan` permite limitar el número de elementos que devuelve en el resultado. Para ello, establezca el parámetro `Limit` en el número máximo de elementos que desea que devuelva la operación `Scan`, antes de la evaluación de la expresión de filtro.

Por ejemplo, suponga que utiliza la operación `Scan` en una tabla con un valor de `Limit` de 6 y sin expresión de filtro. El resultado de `Scan` contiene los seis primeros elementos de la tabla.

Ahora, suponga que agrega una expresión de filtro a `Scan`. En este caso, DynamoDB aplica la expresión de filtro a los seis elementos que se hayan devuelto y descarta los que no coincidan. El resultado de `Scan` final contendrá 6 elementos o menos, según el número de elementos que el filtro elimine.

Paginación de los resultados

DynamoDB pagina los resultados de las operaciones `Scan`. La paginación permite dividir los resultados de `Scan` en "páginas" de datos con un tamaño de 1 MB (o menos). Una aplicación puede procesar la primera página de resultados, luego la segunda página, y así sucesivamente.

Una operación `Scan` única devuelve solamente un conjunto de resultados que se ajuste al límite de tamaño de 1 MB.

Para determinar si hay más resultados y para recuperarlos de página en página, las aplicaciones deben hacer lo siguiente:

1. Examinar el resultado de `Scan` de bajo nivel:
   - Si el resultado contiene una entrada `LastEvaluatedKey`, vaya al paso 2.
   - Si no figura `LastEvaluatedKey` en el resultado, no hay más elementos que recuperar.
2. Crear una nueva solicitud `Scan` con los mismos parámetros que la anterior. Sin embargo, esta vez, se toma el valor `LastEvaluatedKey` del paso 1 y se usa como parámetro `ExclusiveStartKey` en la nueva solicitud `Scan`.
3. Ejecutar la nueva solicitud `Scan`.
4. Ir al paso 1.

Es decir, el valor de `LastEvaluatedKey` de la respuesta de `Scan` debe utilizarse como valor de `ExclusiveStartKey` en la siguiente solicitud `Scan`. Si no hay un componente `LastEvaluatedKey` en una respuesta de `Scan`, significa que se ha recuperado la última página de resultados. La ausencia de `LastEvaluatedKey` es la única forma de saber que se ha alcanzado el final del conjunto de resultados.

Puede utilizar la AWS CLI para ver este comportamiento. La AWS CLI envía solicitudes `Scan` de bajo nivel a DynamoDB una y otra vez hasta que `LastEvaluatedKey` ya no esté presente en los resultados. Considere el siguiente ejemplo de la AWS CLI que examina toda la tabla `Movies`, pero solo devuelve las películas de un determinado género.

```
aws dynamodb scan \
    --table-name Movies \
    --projection-expression "title" \
    --filter-expression 'contains(info.genres,:gen)' \
    --expression-attribute-values '{":gen":{"S":"Sci-Fi"}}' \
    --page-size 100  \
    --debug
```

Normalmente, la AWS CLI se encarga de la paginación automáticamente. No obstante, en este ejemplo, el parámetro AWS CLI de la `--page-size` limita el número de elementos por página. El parámetro `--debug` muestra información de bajo nivel de las solicitudes y las respuestas.

> **Note**
>
> Los resultados de paginación también diferirán en función de los parámetros de entrada que pase.
>
> - El uso de `aws dynamodb scan --table-name Prices --max-items 1` devuelve un `NextToken`
> - El uso de `aws dynamodb scan --table-name Prices --limit 1` devuelve un `LastEvaluatedKey`.
>
> También tenga en cuenta que usar un `--starting-token` en particular requiere el valor de `NextToken`.

Si ejecuta el ejemplo, la primera respuesta de DynamoDB será similar a la siguiente.

```
2017-07-07 12:19:14,389 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":7,"Items":[{"title":{"S":"Monster on the Campus"}},{"title":{"S":"+1"}},
{"title":{"S":"100 Degrees Below Zero"}},{"title":{"S":"About Time"}},{"title":
{"S":"After Earth"}},
{"title":{"S":"Age of Dinosaurs"}},{"title":{"S":"Cloudy with a Chance of Meatballs
 2"}}],
"LastEvaluatedKey":{"year":{"N":"2013"},"title":{"S":"Curse of
 Chucky"}},"ScannedCount":100}'
```

El elemento `LastEvaluatedKey` de la respuesta indica que no se han recuperado todos los elementos. La AWS CLI envía entonces otra solicitud `Scan` a DynamoDB. Este patrón de solicitud y respuesta continúa hasta la respuesta final.

```
2017-07-07 12:19:17,830 - MainThread - botocore.parsers - DEBUG - Response body:
b'{"Count":1,"Items":[{"title":{"S":"WarGames"}}],"ScannedCount":6}'
```

La ausencia de `LastEvaluatedKey` indica que no hay más elementos que recuperar.

> **Note**
>
> Los SDK de AWS se encargan también de las respuestas de DynamoDB de bajo nivel (incluida la presencia o ausencia de `LastEvaluatedKey`) y proporcionan varias

abstracciones para paginar los resultados de `Scan`. Por ejemplo, la interfaz de documentos de SDK para Java proporciona compatibilidad con `java.util.Iterator` para que pueda examinar los resultados de uno en uno.
Para ver ejemplos de código en diversos lenguajes de programación, consulte la Guía de inicio de Amazon DynamoDB y la documentación del SDK de AWS correspondiente a su lenguaje.

Recuento de los elementos en los resultados

Además de los elementos que coinciden con los criterios, la respuesta de `Scan` contiene las entradas siguientes:

- `ScannedCount`: El número de elementos evaluados, antes de que se aplique cualquier `ScanFilter`. Un valor de `ScannedCount` elevado, con pocos resultados de `Count` o ninguno indica que la operación `Scan` ha sido ineficiente. Si no ha utilizado un filtro en la solicitud, `ScannedCount` es igual que `Count`.
- `Count`: El número de elementos restantes después de aplicar una expresión de filtro (si la hay).

> **Note**
>
> Si no se utiliza una expresión de filtro, `ScannedCount` y `Count` tendrán el mismo valor.

Si el tamaño del conjunto de resultados de `Scan` es mayor que 1 MB, entonces `ScannedCount` y `Count` representarán únicamente un recuento parcial de los elementos totales. Deberá llevar a cabo varias operaciones `Scan` para recuperar todos los resultados (consulte Paginación de los resultados).

Cada respuesta de `Scan` contendrá los valores de `ScannedCount` y `Count` de los elementos que se han procesado en esa solicitud `Scan` concreta. Para obtener totales globales para todas las solicitudes `Scan`, puede llevar un recuento total de `ScannedCount` y `Count`.

Unidades de capacidad que consume el análisis

Puede utilizar la operación `Scan` en cualquier tabla o índice secundario. Las operaciones `Scan` consumen las unidades de capacidad de lectura, como se indica a continuación.

Si **`Scan`** se aplica a...	DynamoDB consume unidades de capacidad de lectura de...
Tabla	La capacidad de lectura aprovisionada de la tabla.
Índice secundario global	La capacidad de lectura aprovisionada del índice.
Índice secundario local	La capacidad de lectura aprovisionada de la tabla base.

> **Note**
>
> Las políticas basadas en recursos no admiten actualmente el acceso entre cuentas para operaciones de análisis de índices secundarios.

De forma predeterminada, una operación `Scan` no devuelve datos sobre la cantidad de capacidad de lectura que consume. Sin embargo, puede especificar el parámetro `ReturnConsumedCapacity` en una solicitud `Scan` para obtener esta información. A continuación se muestran los ajustes válidos de `ReturnConsumedCapacity`:

- `NONE`: no se devuelven datos de capacidad consumida. (Esta es la opción predeterminada.)
- `TOTAL`: la respuesta incluye el número total de unidades de capacidad de lectura consumidas.
- `INDEXES`: la respuesta muestra el número total de unidades de capacidad de lectura consumidas, así como la capacidad consumida de cada tabla e índice a los que se ha obtenido acceso.

DynamoDB calcula el número de unidades de capacidad de lectura consumidas según el número de elementos y el tamaño de dichos elementos, no según la cantidad de datos que se devuelven a una aplicación. Por este motivo, el número de unidades de capacidad consumidas es el mismo si se solicitan todos los atributos (el comportamiento predeterminado) o solo algunos de ellos (mediante una expresión de proyección). El número también es el mismo tanto si se utiliza una expresión de filtro como si no. `Scan`consume una unidad de capacidad de lectura mínima para realizar una lectura altamente coherente por segundo o dos lecturas coherentes posteriores por segundo para un elemento de hasta 4 KB. Para leer un elemento mayor que 4 KB, DynamoDB necesita unidades

de solicitud de lectura adicionales. Con las tablas vacías y las tablas muy grandes con una cantidad escasa de claves de partición, es posible que se cobren algunas RCU adicionales además de la cantidad de datos analizados. Esto cubre el costo de atender la solicitud `Scan`, incluso si no hay datos.

Coherencia de lectura para el análisis

De forma predeterminada, una operación `Scan` lleva a cabo lecturas consistentes finales. Esto significa que los resultados de `Scan` podrían no reflejar los cambios provocados por operaciones `PutItem` o `UpdateItem` realizadas recientemente. Para obtener más información, consulte Coherencia de lectura de DynamoDB.

Si requiere lecturas de consistencia alta desde el momento en que se inicie la operación `Scan`, establezca el parámetro `ConsistentRead` en `true` en la solicitud `Scan`. De este modo, se asegurará de que todas las operaciones de escritura que se han completado antes de iniciar `Scan` se incluyan en la respuesta de `Scan`.

Establecer `ConsistentRead` en `true` puede resultar útil para realizar backup o replicaciones de tablas, combinado con DynamoDB Streams. En primer lugar, se utiliza `Scan` con `ConsistentRead` establecido en true para obtener una copia consistente de los datos de la tabla. Durante la operación `Scan`, DynamoDB Streams registra la actividad de escritura adicional que se produce en la tabla. Una vez que la operación `Scan` ha finalizado, puede aplicar la actividad de escritura de la secuencia a la tabla.

> **Note**
>
> Una operación `Scan` con `ConsistentRead` establecido en `true` consumirá el doble de unidades de capacidad de lectura que si `ConsistentRead` se deja establecido en su valor predeterminado (`false`).

Análisis paralelo

De forma predeterminada, la operación `Scan` procesa los datos secuencialmente. Amazon DynamoDB devuelve los datos a la aplicación en incrementos de 1 MB y una aplicación lleva a cabo operaciones `Scan` adicionales para recuperar la siguiente franja de 1 MB de datos.

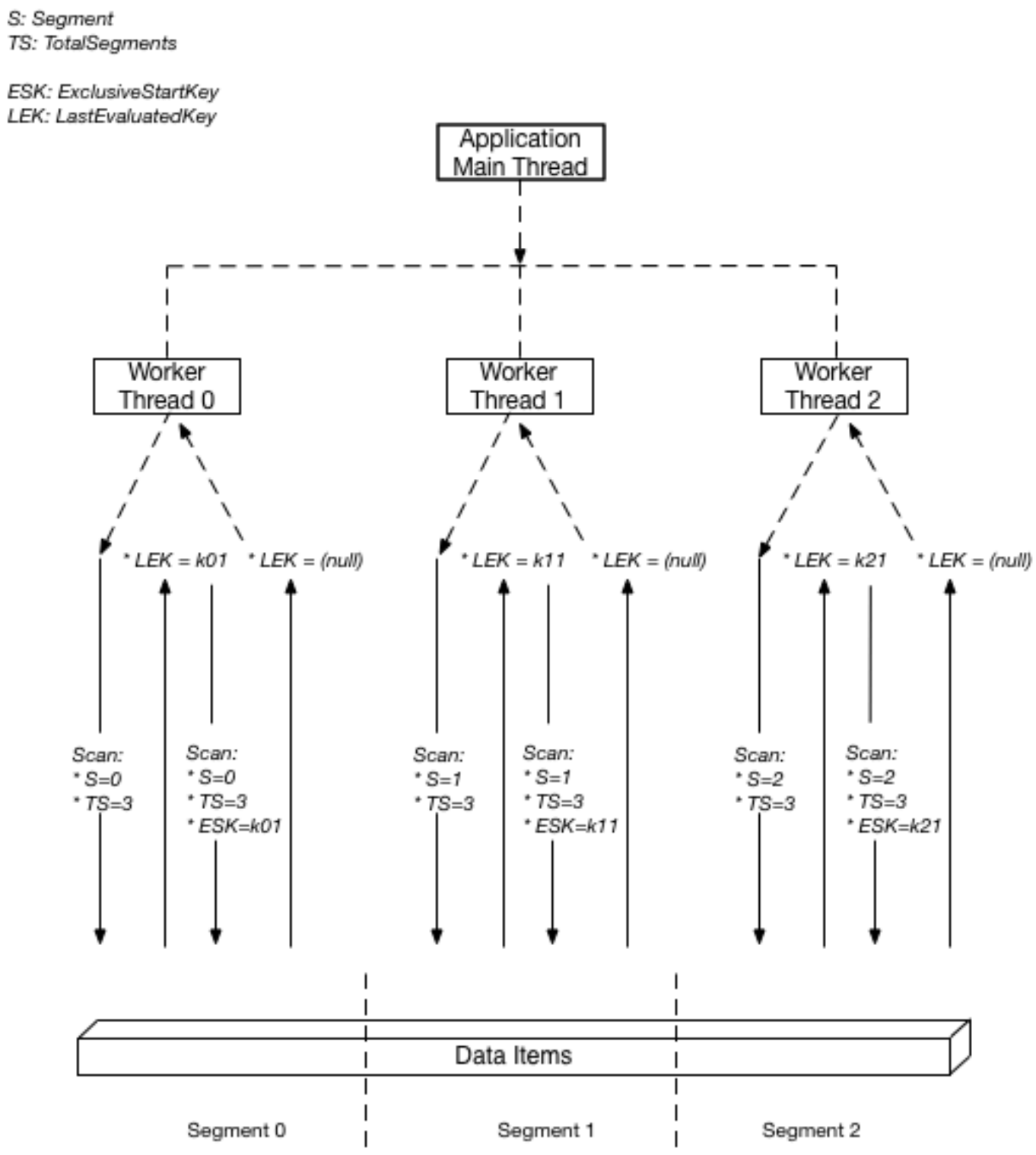
Cuanto mayor es la tabla o el índice que se analiza, más tiempo tarda `Scan` en completarse. Además, puede que una operación `Scan` secuencial no consiga siempre utilizar plenamente la

capacidad de rendimiento de lectura aprovisionada. Esto se debe a que, aunque DynamoDB distribuye los datos de las tablas grandes entre varias particiones físicas, una operación `Scan` solo puede leer una partición a la vez. Por este motivo, el desempeño de una operación `Scan` se ve restringido por el desempeño máximo de cada partición individual.

Para abordar estos problemas, la operación `Scan` puede dividir una tabla o un índice secundario lógicamente en varios segmentos, de tal forma que varios procesos de trabajo de la aplicación puedan examinarlos en paralelo. Cada proceso de trabajo puede ser un subproceso (en los lenguajes de programación que admiten la ejecución de múltiples subprocesos) o un proceso del sistema operativo. Para llevar a cabo un examen en paralelo, cada proceso de trabajo emite su propia solicitud `Scan` con los siguientes parámetros:

- `Segment` segmento que un proceso de trabajo concreto examinará. Cada proceso de trabajo debe utilizar un valor diferente de `Segment`.
- `TotalSegments` número total de segmentos del examen en paralelo. Este valor debe ser el mismo que el número de procesos de trabajo que la aplicación va a utilizar.

En el siguiente diagrama se muestra cómo una aplicación de ejecución multiproceso realiza una operación `Scan` en paralelo con tres grados de paralelismo.



En este diagrama, la aplicación crea tres subprocesos y asigna un número a cada uno de ellos. Los segmentos parten de cero, de modo que el primer número siempre es 0. Cada subproceso emite una solicitud `Scan` y establece el valor de `Segment` en el número designado y el valor de

`TotalSegments`, en 3. Cada subproceso examina su segmento designado, recupera datos en franjas de 1 MB a la vez y devuelve los datos al subproceso principal de la aplicación.

DynamoDB asigna elementos a los segmentos mediante la aplicación de una función hash a la clave de partición de cada elemento. Para un valor de `TotalSegments` determinado, todos los elementos con la misma clave de partición siempre se asignan al mismo `Segment`. Esto significa que en una tabla en la que elemento 1, elemento 2 y elemento 3 comparten `pk="account#123"` (pero tienen claves de clasificación diferentes), el mismo trabajador procesará estos elementos, independientemente de los valores de las claves de clasificación o del tamaño de la recopilación de elementos.

Como la asignación de segmentos se basa únicamente en el hash de la clave de partición, los segmentos pueden distribuirse de forma desigual. Es posible que algunos segmentos no contengan elementos, mientras que otros pueden contener muchas claves de partición con grandes recopilaciones de elementos. Como resultado, aumentar el número total de segmentos no garantiza un rendimiento de análisis más rápido, especialmente cuando las claves de partición no están distribuidas uniformemente en el espacio de claves.

Los valores de `Segment` y `TotalSegments` se aplican a las solicitudes `Scan` individuales; puede utilizar valores diferentes en cualquier momento. Es posible que tenga que experimentar con estos valores y con el número de procesos de trabajo utilizados hasta lograr el desempeño óptimo de la aplicación.

> **Note**
>
> Un examen en paralelo con un gran número de procesos de trabajo puede consumir fácilmente todo el rendimiento aprovisionado de la tabla o el índice que se examina. Es preferible evitar este tipo de exámenes si la tabla o el índice también llevan a cabo una intensa actividad de lectura o escritura de otras aplicaciones.
> Para controlar la cantidad de datos devueltos por cada solicitud, utilice el parámetro `Limit`. Esto puede ayudar a evitar situaciones en las que un proceso de trabajo consume todo el desempeño provisionado a costa de todos los demás procesos de trabajo.

PartiQL: un lenguaje de consulta compatible con SQL para Amazon DynamoDB

Compatibilidad con Amazon DynamoDB PartiQL, un lenguaje de consulta compatible con SQL, para seleccionar, insertar, actualizar y eliminar datos en Amazon DynamoDB. Utilizando PartiQL, puede interactuar fácilmente con tablas de DynamoDB y ejecutar consultas ad hoc utilizando la Consola de administración de AWS, NoSQL Workbench, AWS Command Line Interface y las API de DynamoDB para PartiQL.

Las operaciones PartiQL proporcionan la misma disponibilidad, latencia y rendimiento que las demás operaciones del plano de datos de DynamoDB.

En las siguientes secciones se describe la implementación DynamoDB de PartiQL.

Temas

- ¿Qué es PartiQL?
- PartiQL en Amazon DynamoDB
- Introducción a PartiQL para DynamoDB
- Tipos de datos de PartiQL para DynamoDB
- Instrucciones PartiQL para DynamoDB
- Uso de funciones de PartiQL con DynamoDB
- Operadores aritméticos, comparativos y lógicos de PartiQL para DynamoDB
- Realización de transacciones con PartiQL para DynamoDB
- Ejecución de operaciones por lote con PartiQL para DynamoDB
- Políticas de seguridad de IAM con PartiQL para DynamoDB

¿Qué es PartiQL?

PartiQL proporciona acceso a consultas compatible con SQL en múltiples almacenes de datos que contienen datos estructurados, datos semiestructurados y datos anidados. Es ampliamente utilizado en Amazon y ahora está disponible como parte de muchos servicios de AWS, incluido DynamoDB.

Para obtener la especificación PartiQL y un tutorial sobre el lenguaje de consulta principal, consulte la Documentación de PartiQL.

> **Note**
>
> - Amazon DynamoDB admite subconjunto del lenguaje de consulta de PartiQL.
> - Amazon DynamoDB no admite el formato de datos Amazon Ion o literales de Amazon Ion.

PartiQL en Amazon DynamoDB

Para ejecutar consultas PartiQL en DynamoDB, puede utilizar:

- La consola de DynamoDB
- Uso de NoSQL Workbench
- La AWS Command Line Interface (AWS CLI)
- Las API de DynamoDB

Para obtener información sobre el uso de estos métodos para acceder a DynamoDB, consulte Acceso a DynamoDB.

Introducción a PartiQL para DynamoDB

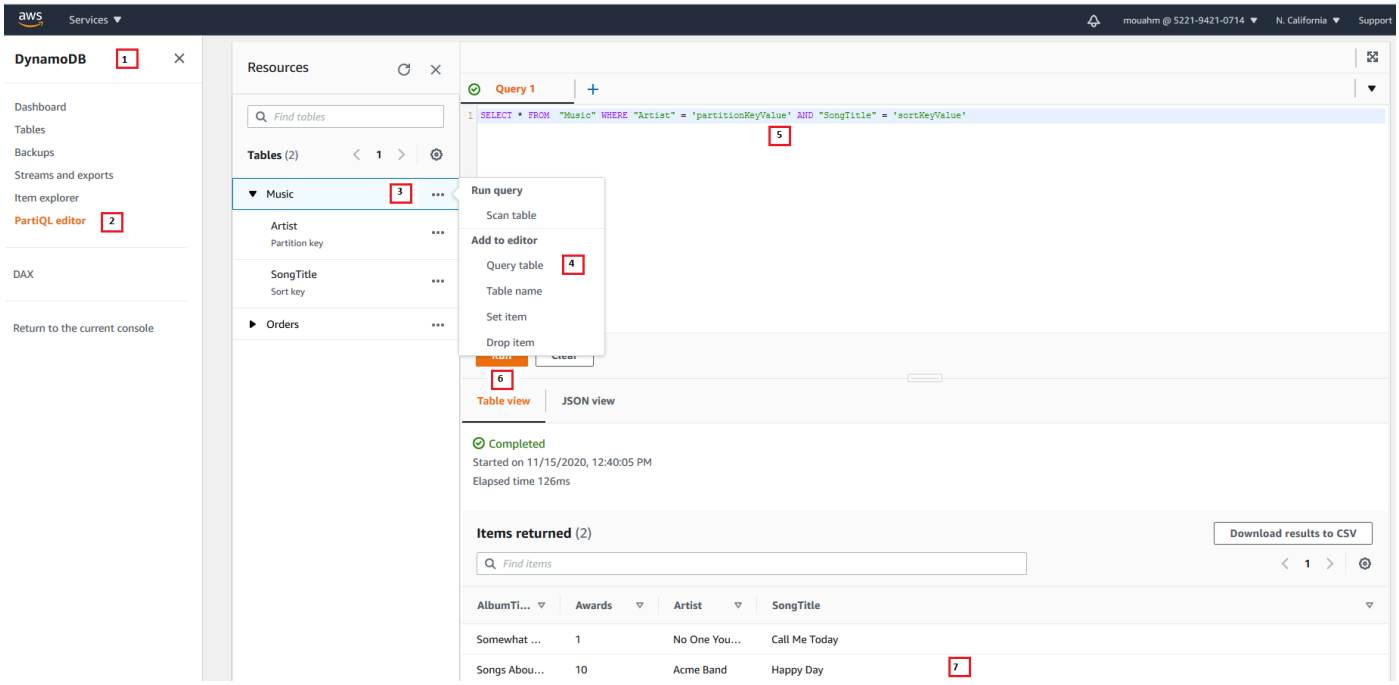
En esta sección se describe cómo utilizar PartiQL para DynamoDB desde la consola de Amazon DynamoDB, la AWS Command Line Interface (AWS CLI) y las API de DynamoDB.

En los ejemplos siguientes, se usa la tabla de DynamoDB que se define en el tutorial Introducción a DynamoDB como requisito previo.

Para obtener información sobre cómo usar la consola de DynamoDB, AWS Command Line Interface o las API de DynamoDB para acceder a DynamoDB, consulte Acceso a DynamoDB.

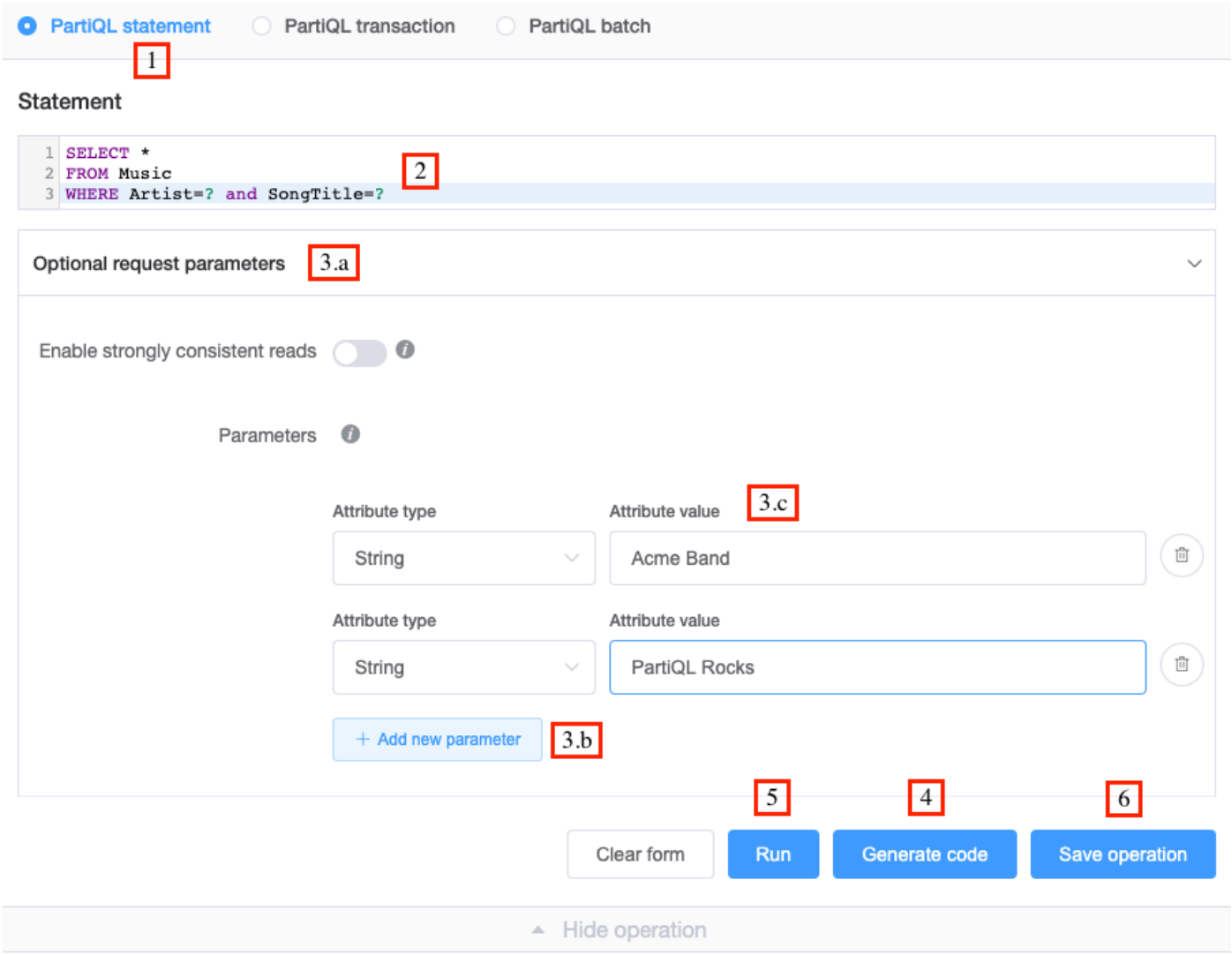
Para descargar y usar el NoSQL Workbench para crear instrucciones PartiQL para DynamoDB elija PartiQL operations (Operaciones de PartiQL) en la esquina superior derecha del NoSQL Workbench para DynamoDB Operation Builder (Creador de operaciones).

Console



1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación del lado izquierdo de la consola, elija Editor PartiQL.
3. Elija la tabla Música.
4. Ellija Tabla de consulta. Esta acción genera una consulta que no dará como resultado un análisis de tabla completo.
5. Reemplazar `partitionKeyValue` con el valor de la cadena `Acme Band`. Reemplazar `sortKeyValue` con el valor de la cadena `Happy Day`.
6. Seleccione el botón Run (Ejecutar).
7. Puede ver los resultados de la consulta seleccionando los botones Table view (Vista de tabla) o JSON view (Vista JSON).

NoSQL workbench



1. Seleccionar Instrucción PartiQL.
2. Ingrese la siguiente PartiQL Instrucción SELECT

```
SELECT *
FROM Music
WHERE Artist=? and SongTitle=?
```

3. Para especificar un valor para los parámetros `Artist` y `SongTitle`:
   a. Elija Parámetros de solicitudes opcionales.
   b. Elija Agregar nuevos parámetros.
   c. Elija el tipo de atributo string y valor `Acme Band`.

d. Repita los pasos b y c y elija el tipo string y valor `PartiQL Rocks`.

4. Si desea generar código, seleccione Generate code (Generar código).

   Seleccione el idioma que desee en las pestañas mostradas. Ahora puede copiar este código y utilizarlo en su aplicación.

5. Si desea que la operación se ejecute inmediatamente, elija Ejecutar.

AWS CLI

1. Cree un elemento en la tabla `Music` utilizando la instrucción INSERT PartiQL.

```
aws dynamodb execute-statement --statement "INSERT INTO Music  \
         VALUE  \
         {'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"
```

2. Recuperar un elemento de la tabla Música mediante la instrucción SELECT PartiQL.

```
aws dynamodb execute-statement --statement "SELECT * FROM Music   \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

3. Actualice un elemento en la tabla `Music` usando la instrucción UPDATE PartiQL.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardsWon=1  \
                                            SET AwardDetail={'Grammys':[2020,
 2018]}  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

   Agregue un valor de lista para un elemento en la tabla `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.Grammys
 =list_append(AwardDetail.Grammys,[2016])  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

   Quitar un valor de lista para un elemento en la tabla `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            REMOVE AwardDetail.Grammys[2]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Agregue un nuevo miembro de mapa para un elemento la tabla `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET AwardDetail.BillBoard=[2020]  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Agregue un nuevo atributo de conjunto de cadenas para un elemento en la tabla `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers =<<'member1',
 'member2'>>  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

Actualice un atributo de conjunto de cadenas para un elemento en la tabla `Music`.

```
aws dynamodb execute-statement --statement "UPDATE Music  \
                                            SET BandMembers
 =set_add(BandMembers, <<'newmember'>>)  \
                                            WHERE Artist='Acme Band' AND
 SongTitle='PartiQL Rocks'"
```

4. Elimine un elemento de la tabla `Music` mediante la instrucción DELETE PartiQL.

```
aws dynamodb execute-statement --statement "DELETE  FROM Music  \
    WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'"
```

Java

```
import java.util.ArrayList;
import java.util.List;

import com.amazonaws.AmazonClientException;
```

```
import com.amazonaws.AmazonServiceException;
import software.amazon.dynamodb.AmazonDynamoDB;
import software.amazon.dynamodb.AmazonDynamoDBClientBuilder;
import software.amazon.dynamodb.model.AttributeValue;
import software.amazon.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.dynamodb.model.ExecuteStatementRequest;
import software.amazon.dynamodb.model.ExecuteStatementResult;
import software.amazon.dynamodb.model.InternalServerErrorException;
import software.amazon.dynamodb.model.ItemCollectionSizeLimitExceededException;
import software.amazon.dynamodb.model.ProvisionedThroughputExceededException;
import software.amazon.dynamodb.model.RequestLimitExceededException;
import software.amazon.dynamodb.model.ResourceNotFoundException;
import software.amazon.dynamodb.model.TransactionConflictException;

public class DynamoDBPartiQGettingStarted {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-1");

        try {
            // Create ExecuteStatementRequest
            ExecuteStatementRequest executeStatementRequest = new
 ExecuteStatementRequest();
            List<AttributeValue> parameters= getPartiQLParameters();

            //Create an item in the Music table using the INSERT PartiQL statement
            processResults(executeStatementRequest(dynamoDB, "INSERT INTO Music
 value {'Artist':?,'SongTitle':?}", parameters));

            //Retrieve an item from the Music table using the SELECT PartiQL
 statement.
            processResults(executeStatementRequest(dynamoDB, "SELECT * FROM Music
 where Artist=? and SongTitle=?", parameters));

            //Update an item in the Music table using the UPDATE PartiQL statement.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music
 SET AwardsWon=1 SET AwardDetail={'Grammys':[2020, 2018]}  where Artist=? and
 SongTitle=?", parameters));

            //Add a list value for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music SET
 AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])  where Artist=? and
 SongTitle=?", parameters));
```

```
            //Remove a list value for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music REMOVE
AwardDetail.Grammys[2]   where Artist=? and SongTitle=?", parameters));

            //Add a new map member for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music set
AwardDetail.BillBoard=[2020] where Artist=? and SongTitle=?", parameters));

            //Add a new string set attribute for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music
SET BandMembers =<<'member1', 'member2'>> where Artist=? and SongTitle=?",
parameters));

            //update a string set attribute for an item in the Music table.
            processResults(executeStatementRequest(dynamoDB, "UPDATE Music SET
BandMembers =set_add(BandMembers, <<'newmember'>>) where Artist=? and SongTitle=?",
parameters));

            //Retrieve an item from the Music table using the SELECT PartiQL
statement.
            processResults(executeStatementRequest(dynamoDB, "SELECT * FROM Music
where Artist=? and SongTitle=?", parameters));

            //delete an item from the Music Table
            processResults(executeStatementRequest(dynamoDB, "DELETE  FROM Music
where Artist=? and SongTitle=?", parameters));
        } catch (Exception e) {
            handleExecuteStatementErrors(e);
        }
    }

    private static AmazonDynamoDB createDynamoDbClient(String region) {
        return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
    }

    private static List<AttributeValue> getPartiQLParameters() {
        List<AttributeValue> parameters = new ArrayList<AttributeValue>();
        parameters.add(new AttributeValue("Acme Band"));
        parameters.add(new AttributeValue("PartiQL Rocks"));
        return parameters;
    }
```

```
   private static ExecuteStatementResult executeStatementRequest(AmazonDynamoDB
client, String statement, List<AttributeValue> parameters ) {
       ExecuteStatementRequest request = new ExecuteStatementRequest();
       request.setStatement(statement);
       request.setParameters(parameters);
       return client.executeStatement(request);
   }

   private static void processResults(ExecuteStatementResult
executeStatementResult) {
       System.out.println("ExecuteStatement successful: "+
executeStatementResult.toString());

   }

   // Handles errors during ExecuteStatement execution. Use recommendations in
error messages below to add error handling specific to
   // your application use-case.
   private static void handleExecuteStatementErrors(Exception exception) {
       try {
           throw exception;
       } catch (ConditionalCheckFailedException ccfe) {
           System.out.println("Condition check specified in the operation failed,
review and update the condition " +
                                      "check before retrying. Error: " +
ccfe.getErrorMessage());
       } catch (TransactionConflictException tce) {
           System.out.println("Operation was rejected because there is an ongoing
transaction for the item, generally " +
                                      "safe to retry with exponential back-off.
Error: " + tce.getErrorMessage());
       } catch (ItemCollectionSizeLimitExceededException icslee) {
           System.out.println("An item collection is too large, you\'re using Local
Secondary Index and exceeded " +
                                      "size limit of items per
partition key. Consider using Global Secondary Index instead. Error: " +
icslee.getErrorMessage());
       } catch (Exception e) {
           handleCommonErrors(e);
       }
   }

   private static void handleCommonErrors(Exception exception) {
       try {
```

```
            throw exception;
        } catch (InternalServerErrorException isee) {
            System.out.println("Internal Server Error, generally safe to retry with
 exponential back-off. Error: " + isee.getErrorMessage());
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for
 your account, increase account level throughput before " +
                                       "retrying. Error: " +
 rlee.getErrorMessage());
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom
 retry strategy make sure to retry with exponential back-off. " +
                                       "Otherwise consider reducing frequency of
 requests or increasing provisioned capacity for your table or secondary index.
 Error: " +
                                       ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists
 before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
            System.out.println("An AmazonServiceException occurred, indicates that
 the request was correctly transmitted to the DynamoDB " +
                                       "service, but for some reason, the service
 was not able to process it, and returned an error response instead. Investigate and
 " +
                                       "configure retry strategy. Error type: " +
 ase.getErrorType() + ". Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that
 the client was unable to get a response from DynamoDB " +
                                       "service, or the client was unable to parse
 the response from the service. Investigate and configure retry strategy. "+
                                       "Error: " + ace.getMessage());
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure
 retry strategy. Error: " + e.getMessage());
        }
    }

}
```

Tipos de datos de PartiQL para DynamoDB

En la tabla siguiente se enumeran los tipos de datos que puede usar con PartiQL para DynamoDB.

Tipo de dato de DynamoDB	Representación PartiQL	Notas
`Boolean`	TRUE, FALSE	No distingue entre mayúsculas y minúsculas.
`Binary`	N/A	Sólo se admite a través de código.
`List`	[valor1, valor2,...]	No hay ninguna restricción respecto a los tipos de datos que se pueden almacenar en una entrada de lista y no es preciso que las entradas de una entrada de lista sean del mismo tipo.
`Map`	{“nombre”: valor}	No hay ninguna restricción respecto a los tipos de datos que se pueden almacenar en una entrada de mapa y no es preciso que las entradas de un mapa sean del mismo tipo.
`Null`	NULL	No distingue entre mayúsculas y minúsculas.
`Number`	1, 1.0, 1e0	Los números pueden ser positivos, negativos o cero. Los números pueden tener hasta 38 dígitos de precisión.
`Number Set`	<<number1, number2>>	Los elementos de un conjunto de números deben ser del tipo Número.

Tipo de dato de DynamoDB	Representación PartiQL	Notas
`String Set`	<<”string1”, “string2”>>	Los elementos de un conjunto de cadenas deben ser de tipo String.
`String`	“Valor de cadena”	Se deben usar comillas simples para especificar valores String.

Ejemplos

La instrucción siguiente muestra cómo insertar los siguientes tipos de datos: `String`, `Number`, `Map`, `List`, `Number Set` y `String Set`.

```
INSERT INTO TypesTable value {'primarykey':'1',
'NumberType':1,
'MapType' : {'entryname1': 'value', 'entryname2': 4},
'ListType': [1,'stringval'],
'NumberSetType':<<1,34,32,4.5>>,
'StringSetType':<<'stringval','stringval2'>>
}
```

La siguiente instrucción muestra cómo insertar nuevos elementos en los tipos `Map`, `List`, `Number Set` y `String Set` y cambiar el valor de un tipo `Number`.

```
UPDATE TypesTable
SET NumberType=NumberType + 100
SET MapType.NewMapEntry=[2020, 'stringvalue', 2.4]
SET ListType = LIST_APPEND(ListType, [4, <<'string1', 'string2'>>])
SET NumberSetType= SET_ADD(NumberSetType, <<345, 48.4>>)
SET StringSetType = SET_ADD(StringSetType, <<'stringsetvalue1', 'stringsetvalue2'>>)
WHERE primarykey='1'
```

La siguiente instrucción muestra cómo eliminar elementos de los tipos `Map`, `List`, `Number Set` y `String Set` y cambiar el valor de un tipo `Number`.

```
UPDATE TypesTable
SET NumberType=NumberType - 1
```

```
REMOVE ListType[1]
REMOVE MapType.NewMapEntry
SET NumberSetType = SET_DELETE( NumberSetType, <<345>>)
SET StringSetType = SET_DELETE( StringSetType, <<'stringsetvalue1'>>)
WHERE primarykey='1'
```

Para obtener más información, consulte Tipos de datos de DynamoDB.

Instrucciones PartiQL para DynamoDB

Amazon DynamoDB admite las siguientes instrucciones PartiQL.

> **Note**
>
> DynamoDB no es compatible con todas las instrucciones PartiQL.
> Esta referencia proporciona sintaxis básica y ejemplos de uso de instrucciones PartiQL que se ejecutan manualmente mediante la AWS CLI o API.

Lenguaje de manipulación de datos(DML) es el conjunto de instrucciones PartiQL que se utiliza para administrar datos en tablas de DynamoDB. Puede usar instrucciones DML para agregar, modificar o eliminar datos de una tabla.

Se admiten las siguientes instrucciones DML y lenguaje de consulta:

- Instrucciones de selección de PartiQL para DynamoDB
- Instrucciones de actualización de PartiQL para DynamoDB
- Instrucciones de inserción de PartiQL para DynamoDB
- Instrucciones de eliminación de PartiQL para DynamoDB

Realización de transacciones con PartiQL para DynamoDB y Ejecución de operaciones por lote con PartiQL para DynamoDB también son compatibles con PartiQL para DynamoDB.

Instrucciones de selección de PartiQL para DynamoDB

Se utiliza la instrucción `SELECT` para recuperar datos de una tabla de Amazon DynamoDB.

Si se usa la instrucción `SELECT` se puede generar un análisis completo de la tabla si no se proporciona una condición de igualdad o IN con una clave de partición en la cláusula WHERE. La

operación de análisis examina cada elemento para comprobar si presenta los valores solicitados y permite utilizar el rendimiento aprovisionado para una tabla o un índice grandes en una sola operación.

Si desea evitar el análisis completo de la tabla en PartiQL, puede:

- Cree su instrucción SELECT para que no resulten en análisis completos de la tabla asegurándose de que su condición de la cláusula WHERE se configura en consecuencia.
- Desactive los análisis completos de tablas mediante la política de IAM especificada en Ejemplo: permitir instrucciones Select y denegar instrucciones de análisis de tabla completa en PartiQL para DynamoDB, en la guía para desarrolladores de DynamoDB.

Para obtener más información, consulte Prácticas recomendadas para consultar y examinar datos, en la guía para desarrolladores de DynamoDB.

Temas

- Sintaxis
- Parameters
- Ejemplos

Sintaxis

```
SELECT expression  [, ...]
FROM table[.index]
[ WHERE condition ] [ [ORDER BY key [DESC|ASC] , ...]
```

Parameters

expression

(Requerido) Una proyección formada a partir del comodín * o una lista de proyección de uno o más nombres de atributos o rutas de documentos del conjunto de resultados. Una expresión puede consistir en llamadas a Uso de funciones de PartiQL con DynamoDB o campos modificados por Operadores aritméticos, comparativos y lógicos de PartiQL para DynamoDB .

table

(Necesario) Nombre de la tabla que se va a consultar.

índice

(Opcional) El nombre del índice que se consultará.

> **Note**
>
> Debe agregar comillas dobles al nombre de la tabla y al nombre del índice al consultar uno.
>
> ```
> SELECT *
> FROM "TableName"."IndexName"
> ```

condition

(Opcional) Criterios de selección para la consulta.

> **Important**
>
> Para asegurarse de que una instrucción `SELECT` no da como resultado un análisis completo de la tabla, la condición de cláusula `WHERE` debe especificar una clave de partición. Utilice el operador de igualdad o IN.
> Por ejemplo, si tiene una tabla `Orders` con una clave de partición `OrderID`y otros atributos que no son clave, incluido una `Address`, las siguientes instrucciones no resultarían en un análisis completo de la tabla:
>
> ```
> SELECT *
> FROM "Orders"
> WHERE OrderID = 100
>
> SELECT *
> FROM "Orders"
> WHERE OrderID = 100 and Address='some address'
>
> SELECT *
> FROM "Orders"
> WHERE OrderID = 100 or OrderID = 200
>
> SELECT *
> FROM "Orders"
> ```

```
WHERE OrderID IN [100, 300, 234]
```

Las siguientes instrucciones de SELECT, sin embargo, resultarán en un análisis completo de la tabla:

```
SELECT *
FROM "Orders"
WHERE OrderID > 1

SELECT *
FROM "Orders"
WHERE Address='some address'

SELECT *
FROM "Orders"
WHERE OrderID = 100 OR Address='some address'
```

clave

(Opcional) Una clave hash o una clave de ordenación que se va a utilizar para ordenar los resultados devueltos. El orden predeterminado es ascendente (ASC) especifique DESC si desea que los resultados se vuelvan a ejecutar en orden descendente.

> **Note**
>
> Si omite la cláusula WHERE, se recuperarán todos los elementos de la tabla.

Ejemplos

La siguiente consulta devuelve un elemento, si existe, de la tabla `Orders` especificando la clave de partición, `OrderID` y utilizando el operador de igualdad.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1
```

La siguiente consulta devuelve todos los elementos de la tabla `Orders` que tiene una clave de partición determinada, `OrderID`, valores utilizando el operador OR.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID = 1 OR OrderID = 2
```

La siguiente consulta devuelve todos los elementos de la tabla `Orders` que tiene una clave de partición determinada, `OrderID`, valores utilizando el operador IN. Los resultados devueltos están en orden descendente, basados en el valor del atributo clave `OrderID`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE OrderID IN [1, 2, 3] ORDER BY OrderID DESC
```

La siguiente consulta muestra un análisis de tabla completo que devuelve todos los elementos de la tabla `Orders` que tienen un `Total` mayor que 500, donde `Total` es un atributo que no es clave.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total > 500
```

La siguiente consulta muestra un análisis de tabla completo que devuelve todos los elementos de la tabla `Orders` con un rango de orden `Total`, utilizando el operador IN y un atributo que no es clave `Total`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total IN [500, 600]
```

La siguiente consulta muestra un análisis de tabla completo que devuelve todos los elementos de la tabla `Orders` dentro de un rango de orden `Total` específico, utilizando el operador BETWEEN y un atributo que no es clave `Total`.

```
SELECT OrderID, Total
FROM "Orders"
WHERE Total BETWEEN 500 AND 600
```

La siguiente consulta devuelve la primera fecha en que se utilizó un dispositivo firestick para ver especificando la clave de partición `CustomerID` y la clave de ordenación `MovieID` en la condición de cláusula WHERE y utilizando rutas de documento en la cláusula SELECT.

```
SELECT Devices.FireStick.DateWatched[0]
FROM WatchList
WHERE CustomerID= 'C1' AND MovieID= 'M1'
```

La siguiente consulta muestra un análisis de tabla completo que devuelve la lista de elementos en los que un dispositivo de firestick se utilizó por primera vez después del 24/12/19 mediante rutas de documento en la condición de cláusula WHERE.

```
SELECT Devices
FROM WatchList
WHERE Devices.FireStick.DateWatched[0] >= '12/24/19'
```

Instrucciones de actualización de PartiQL para DynamoDB

Use la instrucción `UPDATE` para modificar el valor de uno o más atributos dentro de un elemento de una tabla de Amazon DynamoDB.

> **Note**
>
> Sólo puede actualizar un elemento a la vez; no puede emitir una sola instrucción de DynamoDB PartiQL que actualice varios elementos. Para obtener información sobre cómo actualizar varios elementos, consulte Realización de transacciones con PartiQL para DynamoDB or Ejecución de operaciones por lote con PartiQL para DynamoDB.

Temas

- Sintaxis
- Parameters
- Valor devuelto
- Ejemplos

Sintaxis

```
UPDATE  table
```

```
[SET | REMOVE]  path  [=  data] […]
WHERE condition [RETURNING returnvalues]
<returnvalues>  ::= [ALL OLD | MODIFIED OLD | ALL NEW | MODIFIED NEW] *
```

Parameters

table

(Necesario) Tabla que contiene los datos que se van a modificar.

path

(Necesario) Nombre de atributo o ruta de documento que se va a crear o modificar.

data

(Necesario) Valor de atributo o resultado de una operación.

Las operaciones admitidas para usar con SET:

- LIST_APPEND: agrega un valor a un tipo de lista.
- SET_ADD: agrega un valor a un número o conjunto de cadenas.
- SET_DELETE: elimina un valor de un número o conjunto de cadenas.

condition

(Necesario) Criterios de selección para el elemento que se va a modificar. Esta condición debe resolverse con un único valor de clave principal.

returnvalues

(Opcional) Utilizar `returnvalues` si desea obtener los atributos del elemento tal como aparecen antes o después de que se actualicen. Los valores válidos son:

- `ALL OLD *`: devuelve todos los atributos del elemento, tal y como aparecían antes de la operación de actualización.
- `MODIFIED OLD *`: devuelve sólo los atributos actualizados, tal y como aparecían antes de la operación de actualización.
- `ALL NEW *`: devuelve todos los atributos del elemento, tal como aparecen después de la operación de actualización.
- `MODIFIED NEW *`: devuelve solamente los atributos actualizados, tal y como aparecen después de la operación `UpdateItem`.

Valor devuelto

Esta instrucción no devuelve un valor a menos que se especifique el parámetro `returnvalues`.

> **Note**
>
> Si la cláusula WHERE de la instrucción UPDATE no se evalúa como true (verdadero) para ningún elemento de la tabla DynamoDB, se devuelve `ConditionalCheckFailedException`.

Ejemplos

Actualiza el valor del atributo de un elemento existente en. Si el atributo no existe, se crea.

En la siguiente consulta se actualiza un elemento de la tabla `"Music"` agregando un atributo de tipo number (`AwardsWon`) y un atributo de tipo map (`AwardDetail`).

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Puede agregar `RETURNING ALL OLD *` para devolver los atributos tal y como aparecían antes de la operación `Update`.

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL OLD *
```

Devuelve lo siguiente:

```
{
    "Items": [
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
```

```
            }
        }
    ]
}
```

Puede agregar `RETURNING ALL NEW *` para devolver los atributos tal y como aparecían después de la operación `Update`.

```
UPDATE "Music"
SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
RETURNING ALL NEW *
```

Devuelve lo siguiente:

```
{
    "Items": [
        {
            "AwardDetail": {
                "M": {
                    "Grammys": {
                        "L": [
                            {
                                "N": "2020"
                            },
                            {
                                "N": "2018"
                            }
                        ]
                    }
                }
            },
            "AwardsWon": {
                "N": "1"
            }
        }
    ]
}
```

En la siguiente consulta se actualiza un elemento de la tabla `"Music"` agregando a una lista `AwardDetail.Grammys`.

```
UPDATE "Music"
SET AwardDetail.Grammys =list_append(AwardDetail.Grammys,[2016])
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

En la siguiente consulta se actualiza un elemento de la tabla `"Music"` eliminando de una lista `AwardDetail.Grammys`.

```
UPDATE "Music"
REMOVE AwardDetail.Grammys[2]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

En la siguiente consulta se actualiza un elemento de la tabla `"Music"` agregando `BillBoard`al mapa`AwardDetail`.

```
UPDATE "Music"
SET AwardDetail.BillBoard=[2020]
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

En la siguiente consulta se actualiza un elemento de la tabla `"Music"` agregando el atributo de conjunto de string `BandMembers`.

```
UPDATE "Music"
SET BandMembers =<<'member1', 'member2'>>
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

En la siguiente consulta se actualiza un elemento de la tabla `"Music"` agregando `newbandmember` al conjunto de string `BandMembers`.

```
UPDATE "Music"
SET BandMembers =set_add(BandMembers, <<'newbandmember'>>)
WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'
```

Instrucciones de eliminación de PartiQL para DynamoDB

Usar la instrucción `DELETE` para eliminar un elemento existente de la tabla de Amazon DynamoDB.

> **Note**
>
> Solo puede eliminar de a un elemento a la vez. No puede emitir una sola instrucción de DynamoDB PartiQL que elimine varios elementos. Para obtener información sobre

cómo eliminar varios elementos, consulte Realización de transacciones con PartiQL para DynamoDB or Ejecución de operaciones por lote con PartiQL para DynamoDB.

Temas

- Sintaxis
- Parameters
- Valor devuelto
- Ejemplos

Sintaxis

```
DELETE FROM table
 WHERE condition [RETURNING returnvalues]
 <returnvalues>  ::= ALL OLD *
```

Parameters

table

(Necesario) Tabla de DynamoDB que contiene el elemento que se va a eliminar.

condition

(Necesario) Criterios de selección para el elemento que se va a eliminar; esta condición debe resolverse en un único valor de clave principal.

returnvalues

(Opcional) Utilizar `returnvalues` si desea obtener los atributos del elemento tal y como aparecían antes de eliminarlos. Los valores válidos son:

- `ALL OLD *`: se devuelve el contenido del elemento anterior.

Valor devuelto

Esta instrucción no devuelve un valor a menos que se especifique el parámetro `returnvalues`.

> **Note**
>
> Si la tabla de DynamoDB no tiene ningún elemento con la misma clave principal que la del elemento para el que se emite DELETE, se devuelve SUCCESS con 0 elementos eliminados. Si la tabla tiene un elemento con la misma clave principal, pero la condición de la cláusula WHERE de la instrucción DELETE se evalúa como false (falso), se devuelve `ConditionalCheckFailedException`.

Ejemplos

La siguiente consulta elimina un elemento de la tabla `"Music"`.

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
```

Puede agregar el parámetro `RETURNING ALL OLD *` para devolver los datos eliminados.

```
DELETE FROM "Music" WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks'
 RETURNING ALL OLD *
```

La instrucción `Delete` devuelve ahora lo siguiente:

```
{
    "Items": [
        {
            "Artist": {
                "S": "Acme Band"
            },
            "SongTitle": {
                "S": "PartiQL Rocks"
            }
        }
    ]
}
```

Instrucciones de inserción de PartiQL para DynamoDB

Usar la instrucción `INSERT` para agregar un elemento a una tabla de Amazon DynamoDB.

> **Note**
>
> Sólo puede insertar un elemento a la vez; no puede emitir una sola instrucción de DynamoDB PartiQL que inserte varios elementos. Para obtener información sobre cómo insertar varios elementos, consulte Realización de transacciones con PartiQL para DynamoDB or Ejecución de operaciones por lote con PartiQL para DynamoDB.

Temas

- Sintaxis
- Parameters
- Valor devuelto
- Ejemplos

Sintaxis

Inserte un único elemento.

```
INSERT INTO table VALUE item
```

Parameters

table

(Necesario) La tabla en la que desea insertar los datos. La tabla debe existir previamente.

elemento

(Necesario) Un elemento válido de DynamoDB representado como tupla PartiQL. Debe especificar un solo elemento y cada nombre de atributo en el elemento distingue entre mayúsculas y minúsculas y se puede denotar con comillas simples (`'...'`) en PartiQL.

Los valores string también se denotan con comillas simples (`'...'`) en PartiQL.

Valor devuelto

Esta instrucción no devuelve ningún valor.

> **Note**
>
> Si la tabla DynamoDB ya tiene un elemento con la misma clave principal que la clave principal del elemento que se va a insertar, se devuelve `DuplicateItemException`.

Ejemplos

```
INSERT INTO "Music" value {'Artist' : 'Acme Band','SongTitle' : 'PartiQL Rocks'}
```

Uso de funciones de PartiQL con DynamoDB

PartiQL en Amazon DynamoDB admite las siguientes variantes integradas de funciones estándar de SQL.

> **Note**
>
> DynamoDB no admite actualmente ninguna función SQL que no se incluya en esta lista.

Funciones de agregación

- Uso de la función SIZE con PartiQL para Amazon DynamoDB

Funciones condicionales

- Uso de la función EXISTS con PartiQL para DynamoDB
- Uso de la función ATTRIBUTE_TYPE con PartiQL para DynamoDB
- Uso de la función BEGINS_WITH con PartiQL para DynamoDB
- Uso de la función CONTAINS con PartiQL para DynamoDB
- Uso de la función MISSING con PartiQL para DynamoDB

Uso de la función EXISTS con PartiQL para DynamoDB

Puede usar EXISTS para realizar la misma función que`ConditionCheck` hace en la API TransactWriteItems. La función EXISTS sólo se puede utilizar en transacciones.

Dado un valor, devuelve TRUE si el valor es una colección no vacía. De lo contrario, devuelve FALSE.

> **Note**
>
> Esta función sólo se puede utilizar en operaciones transaccionales.

Sintaxis

```
EXISTS ( statement )
```

Argumentos

statement

(Requerido) La instrucción SELECT que la función evalúa.

> **Note**
>
> La instrucción SELECT debe especificar una clave principal completa y otra condición.

Tipo de retorno

`bool`

Ejemplos

```
EXISTS(
    SELECT * FROM "Music"
    WHERE "Artist" = 'Acme Band' AND "SongTitle" = 'PartiQL Rocks')
```

Uso de la función BEGINS_WITH con PartiQL para DynamoDB

Devuelve TRUE si el atributo especificado comienza por una subcadena determinada.

Sintaxis

```
begins_with(path, value )
```

Argumentos

path

(Necesario) Nombre del atributo o ruta del documento que se va a utilizar.

value

(Necesario) La cadena que se va a buscar.

Tipo de retorno

bool

Ejemplos

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND begins_with("Address", '7834 24th')
```

Uso de la función MISSING con PartiQL para DynamoDB

Devuelve TRUE si el elemento no contiene el atributo especificado. Sólo los operadores de igualdad y desigualdad pueden ser utilizados con esta función.

Sintaxis

```
 attributename IS | IS NOT  MISSING
```

Argumentos

attributename

(Necesario) Nombre del atributo que se va a buscar.

Tipo de retorno

bool

Ejemplos

```
SELECT * FROM Music WHERE "Awards" is MISSING
```

Uso de la función ATTRIBUTE_TYPE con PartiQL para DynamoDB

Devuelve TRUE si el atributo de la ruta especificada es de un tipo de datos determinado.

Sintaxis

```
attribute_type( attributename, type )
```

Argumentos

attributename

(Necesario) Nombre del atributo que se va a utilizar.

type

(Necesario) Tipo de atributo que se va a comprobar. Para obtener una lista de valores válidos, consulte attribute_type de DynamoDB.

Tipo de retorno

`bool`

Ejemplos

```
SELECT * FROM "Music" WHERE attribute_type("Artist", 'S')
```

Uso de la función CONTAINS con PartiQL para DynamoDB

Devuelve TRUE si el atributo especificado por la ruta es uno de los siguientes:

- Un valor de tipo String que contiene una determinada subcadena.
- Un valor de tipo Set que contiene una entrada determinada perteneciente al conjunto.

Para obtener más información, consulte la función contains (contiene) de DynamoDB.

Sintaxis

```
contains( path, substring )
```

Argumentos

path

(Necesario) Nombre del atributo o ruta del documento que se va a utilizar.

subcadena

(Necesario) La subcadena de atributo o miembro de conjunto que se va a comprobar. Para obtener más información, consulte la función contains (contiene) de DynamoDB.

Tipo de retorno

`bool`

Ejemplos

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND contains("Address", 'Kirkland')
```

Uso de la función SIZE con PartiQL para Amazon DynamoDB

Devuelve un número que representa el tamaño de un atributo en bytes. A continuación se muestran los tipos de datos válidos para usarlos con size. Para obtener más información, consulte la función size (dimensión) de DynamoDB.

Sintaxis

```
size( path)
```

Argumentos

path

(Necesario) Nombre del atributo o ruta del documento.

Para conocer los tipos admitidos, consulte la función size (dimensión) de DynamoDB.

Tipo de retorno

`int`

Ejemplos

```
SELECT * FROM "Orders" WHERE "OrderID"=1 AND size("Image") >300
```

Operadores aritméticos, comparativos y lógicos de PartiQL para DynamoDB

PartiQL en Amazon DynamoDB admite los siguiente Operadores estándares de SQL.

> **Note**
>
> DynamoDB no admite ningún operador SQL que no se incluya en esta lista.

Operadores aritméticos

Operador	Descripción
+	Add (Suma)
-	Subtract (Sustracción)

Operadores de comparación

Operador	Descripción
=	Igual que
<>	No igual que
!=	No igual que
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

Logical operators (Operadores lógicos)

Operador	Descripción
`AND`	`TRUE` si todas las condiciones separadas por `AND` son `TRUE`
`BETWEEN`	`TRUE` si el operando está en el rango de las comparaciones Este operador incluye los límites inferior y superior de los operandos a los que se aplica.
`IN`	`TRUE` si el operando es igual a uno de una lista de expresiones (con un máximo de 50 valores de atributo hash o un máximo de 100 valores de atributo no clave)
`IS`	`TRUE` si el operando es un tipo de datos PartiQL dado, incluyendo `NULL` o `MISSING`
`NOT`	Invierte el valor de una expresión booleana dada
`OR`	`TRUE` si alguna de las condiciones está separada por `OR` son `TRUE`

Consulte Realización de comparaciones y Evaluaciones lógicas para obtener más información sobre operadores lógicos.

Realización de transacciones con PartiQL para DynamoDB

En esta sección se describe cómo utilizar transacciones con PartiQL para DynamoDB. Las transacciones PartiQL están limitadas a un total de 100 instrucciones (acciones).

Para obtener más información acerca de las transacciones de DynamoDB, consulte Administración de flujos de trabajo complejos con transacciones de DynamoDB.

> **Note**
>
> Toda la transacción debe constar de instrucciones de lectura o de escritura. No puede mezclar ambos en una sola transacción. La función EXISTS es una excepción. Se puede utilizar para verificar la condición de atributos específicos del elemento de una manera similar a `ConditionCheck` en la operación de la API TransactWriteItems.

Temas

- Sintaxis
- Parameters
- Valores devueltos
- Ejemplos

Sintaxis

```
[
   {
      "Statement":" statement ",
      "Parameters":[
         {
            " parametertype " : " parametervalue "
         }, ...]
   } , ...
]
```

Parameters

statement

(Necesario) Una instrucción compatible con PartiQL para DynamoDB.

> **Note**
>
> Toda la transacción debe constar de instrucciones de lectura o de escritura. No puede mezclar ambos en una sola transacción.

parametertype

(Opcional) Tipo DynamoDB, si se utilizaron parámetros al especificar la instrucción PartiQL.

parametervalue

(Opcional) Valor de parámetro si se utilizaron parámetros al especificar la instrucción PartiQL.

Valores devueltos

Esta instrucción no devuelve ningún valor para las operaciones de escritura (INSERT, UPDATE o DELETE). Sin embargo, devuelve valores diferentes para las operaciones de lectura (SELECT) en función de las condiciones especificadas en la cláusula WHERE.

> **Note**
>
> Si alguna de las operaciones singleton INSERT, UPDATE o DELETE devuelve un error, las transacciones se cancelan con la excepción `TransactionCanceledException`, y el código de motivo de cancelación incluye los errores de las operaciones singleton individuales.

Ejemplos

En el ejemplo siguiente, se ejecutan varias instrucciones como transacción.

AWS CLI

1. Guarde el siguiente código JSON en un archivo llamado partiql.json.

```
[
    {
        "Statement": "EXISTS(SELECT * FROM \"Music\" where Artist='No One You
 Know' and SongTitle='Call Me Today' and Awards is  MISSING)"
    },
    {
        "Statement": "INSERT INTO Music value {'Artist':?,'SongTitle':'?'}",
        "Parameters": [{\"S\": \"Acme Band\"}, {\"S\": \"Best Song\"}]
    },
    {
```

```
        "Statement": "UPDATE \"Music\" SET AwardsWon=1 SET
 AwardDetail={'Grammys':[2020, 2018]}  where Artist='Acme Band' and
 SongTitle='PartiQL Rocks'"
    }
]
```

2. Ejecute el comando siguiente en un símbolo del sistema.

```
aws dynamodb execute-transaction --transact-statements  file://partiql.json
```

Java

```
public class DynamoDBPartiqlTransaction {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-2");

        try {
            // Create ExecuteTransactionRequest
            ExecuteTransactionRequest executeTransactionRequest =
 createExecuteTransactionRequest();
            ExecuteTransactionResult executeTransactionResult =
 dynamoDB.executeTransaction(executeTransactionRequest);
            System.out.println("ExecuteTransaction successful.");
            // Handle executeTransactionResult

        } catch (Exception e) {
            handleExecuteTransactionErrors(e);
        }
    }

    private static AmazonDynamoDB createDynamoDbClient(String region) {
        return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
    }

    private static ExecuteTransactionRequest createExecuteTransactionRequest() {
        ExecuteTransactionRequest request = new ExecuteTransactionRequest();

        // Create statements
        List<ParameterizedStatement> statements = getPartiQLTransactionStatements();
```

```
        request.setTransactStatements(statements);
        return request;
    }

    private static List<ParameterizedStatement> getPartiQLTransactionStatements() {
        List<ParameterizedStatement> statements = new
ArrayList<ParameterizedStatement>();

        statements.add(new ParameterizedStatement()
                                .withStatement("EXISTS(SELECT * FROM "Music" where
Artist='No One You Know' and SongTitle='Call Me Today' and Awards is  MISSING)"));

        statements.add(new ParameterizedStatement()
                                .withStatement("INSERT INTO "Music" value
{'Artist':'?','SongTitle':'?'}")
                                .withParameters(new AttributeValue("Acme Band"),new
AttributeValue("Best Song")));

        statements.add(new ParameterizedStatement()
                                .withStatement("UPDATE "Music" SET AwardsWon=1
SET AwardDetail={'Grammys':[2020, 2018]}  where Artist='Acme Band' and
SongTitle='PartiQL Rocks'"));

        return statements;
    }

    // Handles errors during ExecuteTransaction execution. Use recommendations in
error messages below to add error handling specific to
    // your application use-case.
    private static void handleExecuteTransactionErrors(Exception exception) {
        try {
            throw exception;
        } catch (TransactionCanceledException tce) {
            System.out.println("Transaction Cancelled, implies a client issue, fix
before retrying. Error: " + tce.getErrorMessage());
        } catch (TransactionInProgressException tipe) {
            System.out.println("The transaction with the given request token is
already in progress, consider changing " +
                "retry strategy for this type of error. Error: " +
tipe.getErrorMessage());
        } catch (IdempotentParameterMismatchException ipme) {
            System.out.println("Request rejected because it was retried with a
different payload but with a request token that was already used, " +
```

```
                "change request token for this payload to be accepted. Error: " +
ipme.getErrorMessage());
        } catch (Exception e) {
            handleCommonErrors(e);
        }
    }

    private static void handleCommonErrors(Exception exception) {
        try {
            throw exception;
        } catch (InternalServerErrorException isee) {
            System.out.println("Internal Server Error, generally safe to retry with
exponential back-off. Error: " + isee.getErrorMessage());
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for
your account, increase account level throughput before " +
                "retrying. Error: " + rlee.getErrorMessage());
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom
retry strategy make sure to retry with exponential back-off. " +
                "Otherwise consider reducing frequency of requests or increasing
provisioned capacity for your table or secondary index. Error: " +
                ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists
before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
            System.out.println("An AmazonServiceException occurred, indicates that
the request was correctly transmitted to the DynamoDB " +
                "service, but for some reason, the service was not able to process
it, and returned an error response instead. Investigate and " +
                "configure retry strategy. Error type: " + ase.getErrorType() + ".
Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that
the client was unable to get a response from DynamoDB " +
                "service, or the client was unable to parse the response from the
service. Investigate and configure retry strategy. "+
                "Error: " + ace.getMessage());
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure
retry strategy. Error: " + e.getMessage());
        }
    }
```

```
}
```

En el ejemplo siguiente se muestran los distintos valores de retorno cuando DynamoDB lee elementos con condiciones diferentes especificadas en la cláusula WHERE.

AWS CLI

1. Guarde el siguiente código JSON en un archivo llamado partiql.json.

```
[
    // Item exists and projected attribute exists
    {
        "Statement": "SELECT * FROM "Music" WHERE Artist='No One You Know' and
 SongTitle='Call Me Today'"
    },
    // Item exists but projected attributes do not exist
    {
        "Statement": "SELECT non_existent_projected_attribute FROM "Music" WHERE
 Artist='No One You Know' and SongTitle='Call Me Today'"
    },
    // Item does not exist
    {
        "Statement": "SELECT * FROM "Music" WHERE Artist='No One I Know' and
 SongTitle='Call You Today'"
    }
]
```

2. Ejecute el comando siguiente en un símbolo del sistema.

```
aws dynamodb execute-transaction --transact-statements  file://partiql.json
```

3. Se devuelve la siguiente respuesta:

```
{
    "Responses": [
        // Item exists and projected attribute exists
        {
            "Item": {
                "Artist":{
                    "S": "No One You Know"
                },
```

```
                "SongTitle":{
                    "S": "Call Me Today"
                }
            }
        },
        // Item exists but projected attributes do not exist
        {
            "Item": {}
        },
        // Item does not exist
        {}
    ]
}
```

Ejecución de operaciones por lote con PartiQL para DynamoDB

En esta sección se describe cómo utilizar instrucciones por lote con PartiQL para DynamoDB.

> **Note**
>
> - Todo el lote debe constar de instrucciones de lectura o de escritura; no se pueden mezclar ambas en un solo lote.
> - `BatchExecuteStatement` y `BatchWriteItem` no pueden realizar más de 25 instrucciones por lote.
> - `BatchExecuteStatement` hace uso de `BatchGetItem` que toma una lista de claves principales en instrucciones independientes.

Temas

- Sintaxis
- Parameters
- Ejemplos

Sintaxis

```
[
  {
```

```
    "Statement": "SELECT pk FROM ProblemSet WHERE pk = 'p#9StkWHYTxm7x2AqSXcrfu7' AND
 sk = 'info'"
  },
  {
    "Statement": "SELECT pk FROM ProblemSet WHERE pk = 'p#isC2ChceGbxHgESc4szoTE' AND
 sk = 'info'"
  }
]
```

```
[
   {
      "Statement":" statement ",
      "Parameters":[
         {
            " parametertype " : " parametervalue "
         }, ...]
   } , ...
]
```

Parameters

statement

(Necesario) Una instrucción compatible con PartiQL para DynamoDB.

> **Note**
>
> - Todo el lote debe constar de instrucciones de lectura o de escritura; no se pueden mezclar ambas en un solo lote.
> - `BatchExecuteStatement` y `BatchWriteItem` no pueden realizar más de 25 instrucciones por lote.

parametertype

(Opcional) Tipo DynamoDB, si se utilizaron parámetros al especificar la instrucción PartiQL.

parametervalue

(Opcional) Valor de parámetro si se utilizaron parámetros al especificar la instrucción PartiQL.

Ejemplos

AWS CLI

1. Guarde el siguiente json en un archivo llamado partiql.json

```
[
   {
  "Statement": "INSERT INTO Music VALUE {'Artist':?,'SongTitle':?}",
   "Parameters": [{"S": "Acme Band"}, {"S": "Best Song"}]
 },
 {
  "Statement": "UPDATE Music SET AwardsWon=1, AwardDetail={'Grammys':[2020,
 2018]} WHERE Artist='Acme Band' AND SongTitle='PartiQL Rocks'"
    }
]
```

2. Ejecute el comando siguiente en un símbolo del sistema.

```
aws dynamodb batch-execute-statement  --statements  file://partiql.json
```

Java

```
public class DynamoDBPartiqlBatch {

    public static void main(String[] args) {
        // Create the DynamoDB Client with the region you want
        AmazonDynamoDB dynamoDB = createDynamoDbClient("us-west-2");

        try {
            // Create BatchExecuteStatementRequest
            BatchExecuteStatementRequest batchExecuteStatementRequest =
 createBatchExecuteStatementRequest();
            BatchExecuteStatementResult batchExecuteStatementResult =
 dynamoDB.batchExecuteStatement(batchExecuteStatementRequest);
            System.out.println("BatchExecuteStatement successful.");
            // Handle batchExecuteStatementResult

        } catch (Exception e) {
            handleBatchExecuteStatementErrors(e);
        }
    }
```

```
   private static AmazonDynamoDB createDynamoDbClient(String region) {

       return AmazonDynamoDBClientBuilder.standard().withRegion(region).build();
   }

   private static BatchExecuteStatementRequest createBatchExecuteStatementRequest()
{
       BatchExecuteStatementRequest request = new BatchExecuteStatementRequest();

       // Create statements
       List<BatchStatementRequest> statements = getPartiQLBatchStatements();

       request.setStatements(statements);
       return request;
   }

   private static List<BatchStatementRequest> getPartiQLBatchStatements() {
       List<BatchStatementRequest> statements = new
ArrayList<BatchStatementRequest>();

       statements.add(new BatchStatementRequest()
                              .withStatement("INSERT INTO Music value
{'Artist':'Acme Band','SongTitle':'PartiQL Rocks'}"));

       statements.add(new BatchStatementRequest()
                              .withStatement("UPDATE Music set
AwardDetail.BillBoard=[2020] where Artist='Acme Band' and SongTitle='PartiQL
Rocks'"));

       return statements;
   }

   // Handles errors during BatchExecuteStatement execution. Use recommendations in
error messages below to add error handling specific to
   // your application use-case.
   private static void handleBatchExecuteStatementErrors(Exception exception) {
       try {
           throw exception;
       } catch (Exception e) {
           // There are no API specific errors to handle for BatchExecuteStatement,
common DynamoDB API errors are handled below
           handleCommonErrors(e);
       }
```

```
    }

    private static void handleCommonErrors(Exception exception) {
        try {
            throw exception;
        } catch (InternalServerErrorException isee) {
            System.out.println("Internal Server Error, generally safe to retry with
 exponential back-off. Error: " + isee.getErrorMessage());
        } catch (RequestLimitExceededException rlee) {
            System.out.println("Throughput exceeds the current throughput limit for
 your account, increase account level throughput before " +
                "retrying. Error: " + rlee.getErrorMessage());
        } catch (ProvisionedThroughputExceededException ptee) {
            System.out.println("Request rate is too high. If you're using a custom
 retry strategy make sure to retry with exponential back-off. " +
                "Otherwise consider reducing frequency of requests or increasing
 provisioned capacity for your table or secondary index. Error: " +
                ptee.getErrorMessage());
        } catch (ResourceNotFoundException rnfe) {
            System.out.println("One of the tables was not found, verify table exists
 before retrying. Error: " + rnfe.getErrorMessage());
        } catch (AmazonServiceException ase) {
            System.out.println("An AmazonServiceException occurred, indicates that
 the request was correctly transmitted to the DynamoDB " +
                "service, but for some reason, the service was not able to process
 it, and returned an error response instead. Investigate and " +
                "configure retry strategy. Error type: " + ase.getErrorType() + ".
 Error message: " + ase.getErrorMessage());
        } catch (AmazonClientException ace) {
            System.out.println("An AmazonClientException occurred, indicates that
 the client was unable to get a response from DynamoDB " +
                "service, or the client was unable to parse the response from the
 service. Investigate and configure retry strategy. "+
                "Error: " + ace.getMessage());
        } catch (Exception e) {
            System.out.println("An exception occurred, investigate and configure
 retry strategy. Error: " + e.getMessage());
        }
    }

}
```

Políticas de seguridad de IAM con PartiQL para DynamoDB

Los siguientes permisos son necesarios:

- Para leer elementos que utilizan PartiQL para DynamoDB, debe tener el permiso `dynamodb:PartiQLSelect` en la tabla o índice.
- Para insertar elementos utilizando PartiQL para DynamoDB, debe tener el permiso `dynamodb:PartiQLInsert` en la tabla o índice.
- Para actualizar elementos con PartiQL para DynamoDB, debe tener el permiso `dynamodb:PartiQLUpdate` en la tabla o índice.
- Para eliminar elementos que utilizan PartiQL para DynamoDB, debe tener el permiso `dynamodb:PartiQLDelete` en la tabla o índice.

Ejemplo: Permitir todas las instrucciones PartiQL para DynamoDB (Select/Insert/Update/Delete) en una tabla

La siguiente política de IAM concede permisos para ejecutar todas las instrucciones PartiQL para DynamoDB en una tabla.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

Ejemplo: Permitir que PartiQL para DynamoDB seleccione las instrucciones en una tabla

La siguiente política de IAM concede permisos para ejecutar la instrucción `select`sobre una tabla específica.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
      }
   ]
}
```

Ejemplo: Permitir instrucciones de inserción de PartiQL para DynamoDB en un índice

La siguiente política de IAM concede permisos para ejecutar la instrucción `insert` en un índice específico.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music/index/index1"
         ]
```

```
      }
   ]
}
```

Ejemplo: Permitir las instrucciones transaccionales de PartiQL para DynamoDB sólo en una tabla

La siguiente política de IAM concede permisos para ejecutar solamente instrucciones transaccionales en una tabla determinada.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:EnclosingOperation":[
                  "ExecuteTransaction"
               ]
            }
         }
      }
   ]
}
```

Ejemplo: Permitir lecturas y escrituras no transaccionales de PartiQL para DynamoDB y bloquear lecturas y escrituras transaccionales de PartiQL en una tabla.

La siguiente directiva de IAM otorga permisos para ejecutar lecturas y escrituras no transaccionales de PartiQL para DynamoDB a la vez que bloquea lecturas y escrituras transaccionales de PartiQL para DynamoDB.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Deny",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:EnclosingOperation":[
                  "ExecuteTransaction"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLInsert",
            "dynamodb:PartiQLUpdate",
            "dynamodb:PartiQLDelete",
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
         ]
```

```
      }
   ]
}
```

Ejemplo: permitir instrucciones Select y denegar instrucciones de análisis de tabla completa en PartiQL para DynamoDB

La siguiente política de IAM concede permisos para ejecutar la instrucción `select` en una tabla específica mientras se bloquean las instrucciones `select` que dan como resultado un examen completo de toda la tabla.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Deny",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/WatchList"
         ],
         "Condition":{
            "Bool":{
               "dynamodb:FullTableScan":[
                  "true"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "dynamodb:PartiQLSelect"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/WatchList"
         ]
      }
```

```
    ]
}
```

Uso de elementos: Java

Puede usar la API de documentos del AWS SDK para Java para realizar operaciones típicas de creación, lectura, actualización y eliminación (CRUD, por sus siglas en inglés) en los elementos de una tabla de Amazon DynamoDB.

> **Note**
>
> Además, SDK para Java proporciona un modelo de persistencia de objetos, que le permite mapear las clases del lado del cliente a las tablas de DynamoDB. Este enfoque puede reducir la cantidad de código que hay que escribir. Para obtener más información, consulte Java 1.x: DynamoDBMapper.

Esta sección contiene ejemplos de Java que permiten realizar diversas acciones con elementos de la API de documentos de Java y varios ejemplos de trabajo completos.

Temas

- Colocación de un elemento
- Obtención de un elemento
- Escritura por lotes: colocación y eliminación de varios elementos
- Obtención por lotes: obtención de varios elementos
- Actualización de un elemento
- Eliminación de un elemento
- Ejemplo: operaciones CRUD mediante la API de documentos de AWS SDK para Java
- Ejemplo: operaciones por lotes mediante la API de documentos de AWS SDK para Java
- Ejemplo: control de atributos de tipo binario mediante la API de documentos de AWS SDK para Java

Colocación de un elemento

El método `putItem` almacena un elemento en una tabla. Si el elemento existe, sustituye el elemento completo. En lugar de ello, si prefiere actualizar solamente algunos atributos concretos, puede usar el método `updateItem`. Para obtener más información, consulte Actualización de un elemento.

Java v2

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
public class PutItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval> <awards>
 <awardsval> <Songtitle> <songtitleval>

                Where:
                    tableName - The Amazon DynamoDB table in which an item is placed
 (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).
```

```
                    keyval - The key value that represents the item to get (for
example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs
About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
                    SongTitle - The song title (for example, SongTitle).
                    SongTitleVal - The value of the song title (for example, Happy
Day).
                **Warning** This program will  place an item that you specify into a
table!
                """;

        if (args.length != 9) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String albumTitle = args[3];
        String albumTitleValue = args[4];
        String awards = args[5];
        String awardVal = args[6];
        String songTitle = args[7];
        String songTitleVal = args[8];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue,
awards, awardVal, songTitle,
                songTitleVal);
        System.out.println("Done!");
        ddb.close();
    }

    public static void putItemInTable(DynamoDbClient ddb,
            String tableName,
            String key,
```

```
            String keyVal,
            String albumTitle,
            String albumTitleValue,
            String awards,
            String awardVal,
            String songTitle,
            String songTitleVal) {

        HashMap<String, AttributeValue> itemValues = new HashMap<>();
        itemValues.put(key, AttributeValue.builder().s(keyVal).build());
        itemValues.put(songTitle, AttributeValue.builder().s(songTitleVal).build());
        itemValues.put(albumTitle,
 AttributeValue.builder().s(albumTitleValue).build());
        itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();

        try {
            PutItemResponse response = ddb.putItem(request);
            System.out.println(tableName + " was successfully updated. The request
 id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
 name correctly!");
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Java v1

Siga estos pasos:

1. Cree una instancia de la clase DynamoDB.

2. Cree una instancia de la clase `Table` para representar la tabla que desea usar.
3. Cree una instancia de la clase `Item` para representar el nuevo elemento. Debe especificar la clave principal del nuevo elemento y sus atributos.
4. Llame al método `putItem` del objeto `Table` utilizando el `Item` que creó en el paso anterior.

En el siguiente ejemplo de código Java se muestran las tareas anteriores. El código escribe un nuevo elemento en la tabla `ProductCatalog`.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

// Build a list of related items
List<Number> relatedItems = new ArrayList<Number>();
relatedItems.add(341);
relatedItems.add(472);
relatedItems.add(649);

//Build a map of product pictures
Map<String, String> pictures = new HashMap<String, String>();
pictures.put("FrontView", "http://example.com/products/123_front.jpg");
pictures.put("RearView", "http://example.com/products/123_rear.jpg");
pictures.put("SideView", "http://example.com/products/123_left_side.jpg");

//Build a map of product reviews
Map<String, List<String>> reviews = new HashMap<String, List<String>>();

List<String> fiveStarReviews = new ArrayList<String>();
fiveStarReviews.add("Excellent! Can't recommend it highly enough!  Buy it!");
fiveStarReviews.add("Do yourself a favor and buy this");
reviews.put("FiveStar", fiveStarReviews);

List<String> oneStarReviews = new ArrayList<String>();
oneStarReviews.add("Terrible product!  Do not buy this.");
reviews.put("OneStar", oneStarReviews);

// Build the item
Item item = new Item()
    .withPrimaryKey("Id", 123)
```

```
    .withString("Title", "Bicycle 123")
    .withString("Description", "123 description")
    .withString("BicycleType", "Hybrid")
    .withString("Brand", "Brand-Company C")
    .withNumber("Price", 500)
    .withStringSet("Color",  new HashSet<String>(Arrays.asList("Red", "Black")))
    .withString("ProductCategory", "Bicycle")
    .withBoolean("InStock", true)
    .withNull("QuantityOnHand")
    .withList("RelatedItems", relatedItems)
    .withMap("Pictures", pictures)
    .withMap("Reviews", reviews);

// Write the item to the table
PutItemOutcome outcome = table.putItem(item);
```

En el ejemplo anterior, el elemento tiene atributos que son escalares (`String`, `Number`, `Boolean`, `Null`), conjuntos (`String Set`) y tipos de documentos (`List`, `Map`).

Especificación de parámetros opcionales

Junto con los parámetros requeridos, puede especificar también otros opcionales en el método `putItem`. Por ejemplo, en el ejemplo de código Java siguiente se utiliza un parámetro opcional que permite especificar una condición para cargar el elemento. Si la condición especificada no se cumple, AWS SDK para Java genera una excepción `ConditionalCheckFailedException`. En el ejemplo de código se especifican los parámetros opcionales siguientes en el método `putItem`:

- Una expresión `ConditionExpression` que define las condiciones de la solicitud. El código define la condición siguiente: si el elemento existente tiene la misma clave principal, solo se sustituirá si tiene también un atributo ISBN igual a un valor específico.
- Un mapa de `ExpressionAttributeValues` que se usa en la condición. En este caso, solo se requiere una sustitución: el marcador de posición `:val` de la expresión de condición se sustituye en el tiempo de ejecución por el valor de ISBN real que se va a comprobar.

En el siguiente ejemplo se agrega un nuevo elemento de libro utilizando estos parámetros opcionales.

Example

```
Item item = new Item()
```

```
    .withPrimaryKey("Id", 104)
    .withString("Title", "Book 104 Title")
    .withString("ISBN", "444-4444444444")
    .withNumber("Price", 20)
    .withStringSet("Authors",
        new HashSet<String>(Arrays.asList("Author1", "Author2")));

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val", "444-4444444444");

PutItemOutcome outcome = table.putItem(
    item,
    "ISBN = :val", // ConditionExpression parameter
    null,          // ExpressionAttributeNames parameter - we're not using it for this
 example
    expressionAttributeValues);
```

PutItem y documentos JSON

Puede almacenar un documento JSON como atributo en una tabla de DynamoDB. Para ello, se utiliza el método `withJSON` de `Item`. Este método analiza el documento JSON y mapea cada componente a un tipo de datos nativo de DynamoDB.

Suponga que desea almacenar el siguiente documento JSON, que contiene los proveedores que pueden servir pedidos de un producto determinado.

Example

```
{
    "V01": {
        "Name": "Acme Books",
        "Offices": [ "Seattle" ]
    },
    "V02": {
        "Name": "New Publishers, Inc.",
        "Offices": ["London", "New York"
        ]
    },
    "V03": {
        "Name": "Better Buy Books",
        "Offices": [ "Tokyo", "Los Angeles", "Sydney"
        ]
    }
```

```
}
```

Puede utilizar el método `withJSON` para almacenar esta información en la tabla `ProductCatalog` en un atributo de tipo `Map` denominado `VendorInfo`. En el siguiente ejemplo de código Java se muestra cómo hacerlo.

```
// Convert the document into a String.  Must escape all double-quotes.
String vendorDocument = "{"
    + "    \"V01\": {"
    + "        \"Name\": \"Acme Books\","
    + "        \"Offices\": [ \"Seattle\" ]"
    + "    },"
    + "    \"V02\": {"
    + "        \"Name\": \"New Publishers, Inc.\","
    + "        \"Offices\": [ \"London\", \"New York\"" + "]" + "},"
    + "    \"V03\": {"
    + "        \"Name\": \"Better Buy Books\","
    +          "\"Offices\": [ \"Tokyo\", \"Los Angeles\", \"Sydney\""
    + "            ]"
    + "        }"
    + "    }";

Item item = new Item()
    .withPrimaryKey("Id", 210)
    .withString("Title", "Book 210 Title")
    .withString("ISBN", "210-2102102102")
    .withNumber("Price", 30)
    .withJSON("VendorInfo", vendorDocument);

PutItemOutcome outcome = table.putItem(item);
```

Obtención de un elemento

Para recuperar un solo elemento, utilice el método `getItem` de un objeto `Table`. Siga estos pasos:

1. Cree una instancia de la clase `DynamoDB`.
2. Cree una instancia de la clase `Table` para representar la tabla que desea usar.
3. Llame al método `getItem` de la instancia de `Table`. Es preciso especificar la clave principal del elemento que se desea recuperar.

En el siguiente ejemplo de código Java se muestran los pasos anteriores. El código obtiene el elemento que tiene la clave de partición especificada.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

Item item = table.getItem("Id", 210);
```

Especificación de parámetros opcionales

Junto con los parámetros requeridos, puede especificar también otros opcionales del método `getItem`. Por ejemplo, en el siguiente ejemplo de código Java se utiliza un método opcional para recuperar solo una lista concreta de atributos y especificar lecturas de consistencia alta. Para obtener más información sobre la consistencia de lectura, consulte Coherencia de lectura de DynamoDB.

Puede usar una expresión `ProjectionExpression` para recuperar solamente algunos atributos o componentes concretos, en lugar de un elemento completo. Una expresión `ProjectionExpression` permite especificar atributos de nivel superior o anidados mediante rutas de documentos. Para obtener más información, consulte Uso de expresiones de proyección en DynamoDB.

Los parámetros del método `getItem` no permiten especificar la consistencia de .lectura Sin embargo, puede crear una especificación `GetItemSpec`, que proporciona acceso pleno a toda la información de entrada de la operación de bajo nivel `GetItem`. En el ejemplo de código siguiente se crea una especificación `GetItemSpec` y se utiliza como información de entrada para el método `getItem`.

Example

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 206)
    .withProjectionExpression("Id, Title, RelatedItems[0], Reviews.FiveStar")
    .withConsistentRead(true);

Item item = table.getItem(spec);

System.out.println(item.toJSONPretty());
```

Para imprimir un `Item` en formato fácil de leer, utilice el método `toJSONPretty`. El resultado del ejemplo anterior tiene este aspecto.

```
{
  "RelatedItems" : [ 341 ],
  "Reviews" : {
    "FiveStar" : [ "Excellent! Can't recommend it highly enough! Buy it!", "Do yourself
 a favor and buy this" ]
  },
  "Id" : 123,
  "Title" : "20-Bicycle 123"
}
```

GetItem y documentos JSON

En la sección PutItem y documentos JSON, almacena un documento JSON en un atributo `Map` denominado `VendorInfo`. Puede utilizar el método `getItem` para recuperar todo el documento en formato JSON. O puede usar la notación de ruta de documento para recuperar solo algunos de los componentes de ese documento. En el siguiente ejemplo de código Java se muestran estas técnicas.

```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 210);

System.out.println("All vendor info:");
spec.withProjectionExpression("VendorInfo");
System.out.println(table.getItem(spec).toJSON());

System.out.println("A single vendor:");
spec.withProjectionExpression("VendorInfo.V03");
System.out.println(table.getItem(spec).toJSON());

System.out.println("First office location for this vendor:");
spec.withProjectionExpression("VendorInfo.V03.Offices[0]");
System.out.println(table.getItem(spec).toJSON());
```

El resultado del ejemplo anterior tiene este aspecto.

```
All vendor info:
```

```
{"VendorInfo":{"V03":{"Name":"Better Buy Books","Offices":["Tokyo","Los
 Angeles","Sydney"]},"V02":{"Name":"New Publishers, Inc.","Offices":["London","New
 York"]},"V01":{"Name":"Acme Books","Offices":["Seattle"]}}}
A single vendor:
{"VendorInfo":{"V03":{"Name":"Better Buy Books","Offices":["Tokyo","Los
 Angeles","Sydney"]}}}
First office location for a single vendor:
{"VendorInfo":{"V03":{"Offices":["Tokyo"]}}}
```

> **Note**
>
> Puede usar el método `toJSON` para convertir cualquier elemento (o sus atributos) en una cadena con formato JSON. En el siguiente ejemplo de código se recuperan varios atributos de nivel superior y anidados y se imprimen los resultados como JSON.
>
> ```
> GetItemSpec spec = new GetItemSpec()
>     .withPrimaryKey("Id", 210)
>     .withProjectionExpression("VendorInfo.V01, Title, Price");
>
> Item item = table.getItem(spec);
> System.out.println(item.toJSON());
> ```
>
> El resultado es similar al siguiente.
>
> ```
> {"VendorInfo":{"V01":{"Name":"Acme Books","Offices":
> ["Seattle"]}},"Price":30,"Title":"Book 210 Title"}
> ```

Escritura por lotes: colocación y eliminación de varios elementos

La escritura por lotes se refiere a colocar y eliminar varios elementos en un lote. El método `batchWriteItem` permite colocar y eliminar varios elementos de una o varias tablas con una sola llamada. A continuación se indican los pasos para colocar o eliminar varios elementos mediante la API de documentos del AWS SDK para Java.

1. Cree una instancia de la clase `DynamoDB`.
2. Cree una instancia de la clase `TableWriteItems` que describe todas las operaciones de colocación y eliminación de una tabla. Si desea escribir en varias tablas en una misma operación de escritura por lotes, debe crear una instancia de `TableWriteItems` por cada tabla.

3. Llame al método `batchWriteItem` proporcionando los objetos `TableWriteItems` que creó en el paso anterior.
4. Procese la respuesta. Debe comprobar si en la respuesta se ha devuelto algún elemento de solicitud sin procesar. Esto puede ocurrir si se alcanza la cuota de rendimiento aprovisionado o se produce algún otro error transitorio. Además, DynamoDB limita el tamaño de la solicitud y el número de operaciones que se pueden especificar en ella. Si supera estos límites, DynamoDB rechaza la solicitud. Para obtener más información, consulte Cuotas en Amazon DynamoDB.

En el siguiente ejemplo de código Java se muestran los pasos anteriores. El ejemplo lleva a cabo una operación `batchWriteItem` en dos tablas: `Forum` y `Thread`. Los objetos `TableWriteItems` correspondientes definen las siguientes acciones:

- Colocar un elemento en la tabla `Forum`.
- Colocar y eliminar un elemento en la tabla `Thread`.

A continuación, el código llama a `batchWriteItem` para llevar a cabo la operación.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

TableWriteItems forumTableWriteItems = new TableWriteItems("Forum")
    .withItemsToPut(
        new Item()
            .withPrimaryKey("Name", "Amazon RDS")
            .withNumber("Threads", 0));

TableWriteItems threadTableWriteItems = new TableWriteItems("Thread")
    .withItemsToPut(
        new Item()
            .withPrimaryKey("ForumName","Amazon RDS","Subject","Amazon RDS Thread 1")
    .withHashAndRangeKeysToDelete("ForumName","Some partition key value", "Amazon S3",
 "Some sort key value");

BatchWriteItemOutcome outcome = dynamoDB.batchWriteItem(forumTableWriteItems,
 threadTableWriteItems);

// Code for checking unprocessed items is omitted in this example
```

Para ver un ejemplo práctico, consulte Ejemplo: operación de escritura por lotes mediante la API de documentos de AWS SDK para Java.

Obtención por lotes: obtención de varios elementos

El método `batchGetItem` permite recuperar varios elementos de una o varias tablas. Para recuperar un solo elemento, puede usar el método `getItem`.

Siga estos pasos:

1. Cree una instancia de la clase `DynamoDB`.
2. Cree una instancia de la clase `TableKeysAndAttributes` que describe una lista de valores de clave principal que se van a recuperar de una tabla. Si desea leer de varias tablas en una misma operación de obtención por lotes, debe crear una instancia de `TableKeysAndAttributes` por cada tabla.
3. Llame al método `batchGetItem` proporcionando los objetos `TableKeysAndAttributes` que creó en el paso anterior.

En el siguiente ejemplo de código Java se muestran los pasos anteriores. El ejemplo recupera dos elementos de la tabla `Forum` y tres de la tabla `Thread`.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

    TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes(forumTableName);
    forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

TableKeysAndAttributes threadTableKeysAndAttributes = new
 TableKeysAndAttributes(threadTableName);
threadTableKeysAndAttributes.addHashAndRangePrimaryKeys("ForumName", "Subject",
    "Amazon DynamoDB","DynamoDB Thread 1",
    "Amazon DynamoDB","DynamoDB Thread 2",
    "Amazon S3","S3 Thread 1");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(
    forumTableKeysAndAttributes, threadTableKeysAndAttributes);
```

```
for (String tableName : outcome.getTableItems().keySet()) {
    System.out.println("Items in table " + tableName);
    List<Item> items = outcome.getTableItems().get(tableName);
    for (Item item : items) {
        System.out.println(item);
    }
}
```

Especificación de parámetros opcionales

Junto con los parámetros requeridos, puede especificar también otros opcionales cuando use `batchGetItem`. Por ejemplo, puede proporcionar una expresión `ProjectionExpression` con cada `TableKeysAndAttributes` que defina. Esto le permite especificar los atributos que desea recuperar de la tabla.

En el siguiente ejemplo de código se recuperan dos elementos de la tabla `Forum`. El parámetro `withProjectionExpression` especifica que solamente hay que recuperar el atributo `Threads`.

Example

```
TableKeysAndAttributes forumTableKeysAndAttributes = new
 TableKeysAndAttributes("Forum")
    .withProjectionExpression("Threads");

forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name",
    "Amazon S3",
    "Amazon DynamoDB");

BatchGetItemOutcome outcome = dynamoDB.batchGetItem(forumTableKeysAndAttributes);
```

Actualización de un elemento

El método `updateItem` de un objeto `Table` permite actualizar los valores de atributos presentes, agregar atributos nuevos o eliminarlos de un elemento existente.

El método `updateItem` se comporta de la siguiente manera:

- Si un elemento no existe (no hay ningún elemento en la tabla con la clave principal especificada), `updateItem` agrega un elemento nuevo a la tabla.
- Si un elemento ya existe, `updateItem` lleva a cabo la actualización según lo especificado en el parámetro `UpdateExpression`.

> **Note**
>
> También es posible actualizar un elemento mediante `putItem`. Por ejemplo, si llama a `putItem` para agregar un elemento a la tabla pero ya existe uno con la clave principal especificada, `putItem` sustituye el elemento completo. Si hay atributos en el elemento existente que no se especifican en la información de entrada, `putItem` los elimina del elemento.
>
> En general, recomendamos usar `updateItem` siempre que desee modificar atributos de elementos. El método `updateItem` solo modifica los atributos del elemento que especifique en la información de entrada, pero deja los demás atributos del elemento tal cual estaban.

Siga estos pasos:

1. Cree una instancia de la clase `Table` para representar la tabla que desea usar.
2. Llame al método `updateTable` de la instancia de `Table`. Debe especificar la clave principal del elemento que desea recuperar, junto con una expresión `UpdateExpression` que describa los atributos que hay que cambiar y cómo modificarlos.

En el siguiente ejemplo de código Java se muestran las tareas anteriores. El código actualiza un elemento de libro en la tabla `ProductCatalog`. Se agrega un nuevo autor al conjunto `Authors` y se elimina el atributo `ISBN` existente. También se reduce el precio en una unidad.

Se utiliza un mapa `ExpressionAttributeValues` en `UpdateExpression`. Los marcadores de posición `:val1` y `:val2` se sustituyen en tiempo de ejecución por los valores reales de `Authors` y `Price`.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

Map<String, String> expressionAttributeNames = new HashMap<String, String>();
expressionAttributeNames.put("#A", "Authors");
expressionAttributeNames.put("#P", "Price");
expressionAttributeNames.put("#I", "ISBN");

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val1",
```

```
    new HashSet<String>(Arrays.asList("Author YY","Author ZZ")));
expressionAttributeValues.put(":val2", 1);   //Price

UpdateItemOutcome outcome =  table.updateItem(
    "Id",          // key attribute name
    101,           // key attribute value
    "add #A :val1 set #P = #P - :val2 remove #I", // UpdateExpression
    expressionAttributeNames,
    expressionAttributeValues);
```

Especificación de parámetros opcionales

Junto con los parámetros requeridos, puede especificar también parámetros opcionales para el método `updateItem`, incluida una condición que debe cumplirse para que se lleve a cabo la actualización. Si la condición especificada no se cumple, AWS SDK para Java genera una excepción `ConditionalCheckFailedException`. Por ejemplo, el siguiente ejemplo de código Java actualiza de forma condicional el precio de un elemento de libro a 25. Especifica una expresión `ConditionExpression` en la que se indica que el precio solo debe actualizarse si el precio actual es 20.

Example

```
Table table = dynamoDB.getTable("ProductCatalog");

Map<String, String> expressionAttributeNames = new HashMap<String, String>();
expressionAttributeNames.put("#P", "Price");

Map<String, Object> expressionAttributeValues = new HashMap<String, Object>();
expressionAttributeValues.put(":val1", 25);  // update Price to 25...
expressionAttributeValues.put(":val2", 20);  //...but only if existing Price is 20

UpdateItemOutcome outcome = table.updateItem(
    new PrimaryKey("Id",101),
    "set #P = :val1", // UpdateExpression
    "#P = :val2",     // ConditionExpression
    expressionAttributeNames,
    expressionAttributeValues);
```

Contador atómico

Puede usar `updateItem` para implementar un contador atómico y aumentar o reducir el valor de un atributo existente sin interferir con las demás solicitudes de escritura. Para incrementar un contador

atómico, use una expresión `UpdateExpression` con la acción `set` para sumar un valor numérico a una atributo existente de tipo `Number`.

En el siguiente ejemplo se pone en práctica lo anterior y se incrementa el atributo `Quantity` en una unidad. También se demuestra cómo usar el parámetro `ExpressionAttributeNames` en una expresión `UpdateExpression`.

```
Table table = dynamoDB.getTable("ProductCatalog");

Map<String,String> expressionAttributeNames = new HashMap<String,String>();
expressionAttributeNames.put("#p", "PageCount");

Map<String,Object> expressionAttributeValues = new HashMap<String,Object>();
expressionAttributeValues.put(":val", 1);

UpdateItemOutcome outcome = table.updateItem(
    "Id", 121,
    "set #p = #p + :val",
    expressionAttributeNames,
    expressionAttributeValues);
```

Eliminación de un elemento

El método `deleteItem` elimina un elemento de una tabla. Es preciso proporcionar la clave principal del elemento que se desea eliminar.

Siga estos pasos:

1. Cree una instancia del cliente de `DynamoDB`.
2. Llame al método `deleteItem` proporcionando la clave del elemento que desea eliminar.

En el siguiente ejemplo de Java se muestran estas tareas.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("ProductCatalog");

DeleteItemOutcome outcome = table.deleteItem("Id", 101);
```

Especificación de parámetros opcionales

Puede especificar parámetros opcionales para `deleteItem`. Por ejemplo, el siguiente ejemplo de código Java incluye una expresión `ConditionExpression` que indica que un elemento de libro de `ProductCatalog` solo se puede eliminar si el libro está descatalogado (el atributo `InPublication` es false).

Example

```
Map<String,Object> expressionAttributeValues = new HashMap<String,Object>();
expressionAttributeValues.put(":val", false);

DeleteItemOutcome outcome = table.deleteItem("Id",103,
    "InPublication = :val",
    null, // ExpressionAttributeNames - not used in this example
    expressionAttributeValues);
```

Ejemplo: operaciones CRUD mediante la API de documentos de AWS SDK para Java

En el siguiente ejemplo de código se ilustran las operaciones CRUD en un elemento de Amazon DynamoDB. En el ejemplo se crea un elemento, se recupera, se llevan a cabo varias actualizaciones y, por último, se elimina.

> **Note**
>
> Además, SDK para Java proporciona un modelo de persistencia de objetos, que le permite mapear las clases del lado del cliente a las tablas de DynamoDB. Este enfoque puede reducir la cantidad de código que hay que escribir. Para obtener más información, consulte Java 1.x: DynamoDBMapper.

> **Note**
>
> En este ejemplo de código se supone que ya ha cargado datos en DynamoDB para su cuenta siguiendo las instrucciones de la sección Creación de tablas y carga de datos para ejemplos de código en DynamoDB.
> Para obtener instrucciones paso a paso acerca de cómo ejecutar el siguiente ejemplo, consulte Ejemplos de código Java.

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DeleteItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.UpdateItemOutcome;
import com.amazonaws.services.dynamodbv2.document.spec.DeleteItemSpec;
import com.amazonaws.services.dynamodbv2.document.spec.UpdateItemSpec;
import com.amazonaws.services.dynamodbv2.document.utils.NameMap;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.ReturnValue;

public class DocumentAPIItemCRUDExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String tableName = "ProductCatalog";

    public static void main(String[] args) throws IOException {

        createItems();

        retrieveItem();

        // Perform various updates.
        updateMultipleAttributes();
        updateAddNewAttribute();
        updateExistingAttributeConditionally();

        // Delete the item.
        deleteItem();
```

```
    }

    private static void createItems() {

        Table table = dynamoDB.getTable(tableName);
        try {

            Item item = new Item().withPrimaryKey("Id", 120).withString("Title", "Book
120 Title")
                    .withString("ISBN", "120-1111111111")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author12", "Author22")))
                    .withNumber("Price", 20).withString("Dimensions",
"8.5x11.0x.75").withNumber("PageCount", 500)
                    .withBoolean("InPublication", false).withString("ProductCategory",
"Book");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 121).withString("Title", "Book 121
Title")
                    .withString("ISBN", "121-1111111111")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author21", "Author 22")))
                    .withNumber("Price", 20).withString("Dimensions",
"8.5x11.0x.75").withNumber("PageCount", 500)
                    .withBoolean("InPublication", true).withString("ProductCategory",
"Book");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Create items failed.");
            System.err.println(e.getMessage());

        }
    }

    private static void retrieveItem() {
        Table table = dynamoDB.getTable(tableName);

        try {

            Item item = table.getItem("Id", 120, "Id, ISBN, Title, Authors", null);

            System.out.println("Printing item after retrieving it....");
```

```
            System.out.println(item.toJSONPretty());

        } catch (Exception e) {
            System.err.println("GetItem failed.");
            System.err.println(e.getMessage());
        }

    }

    private static void updateAddNewAttribute() {
        Table table = dynamoDB.getTable(tableName);

        try {

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
121)
                    .withUpdateExpression("set #na = :val1").withNameMap(new
NameMap().with("#na", "NewAttribute"))
                    .withValueMap(new ValueMap().withString(":val1", "Some value"))
                    .withReturnValues(ReturnValue.ALL_NEW);

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
            System.out.println("Printing item after adding new attribute...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Failed to add new attribute in " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void updateMultipleAttributes() {

        Table table = dynamoDB.getTable(tableName);

        try {

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
120)
                    .withUpdateExpression("add #a :val1 set #na=:val2")
                    .withNameMap(new NameMap().with("#a", "Authors").with("#na",
"NewAttribute"))
```

```
                    .withValueMap(
                            new ValueMap().withStringSet(":val1", "Author YY", "Author
ZZ").withString(":val2",
                                    "someValue"))
                    .withReturnValues(ReturnValue.ALL_NEW);

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
            System.out.println("Printing item after multiple attribute update...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Failed to update multiple attributes in " + tableName);
            System.err.println(e.getMessage());

        }
    }

    private static void updateExistingAttributeConditionally() {

        Table table = dynamoDB.getTable(tableName);

        try {

            // Specify the desired price (25.00) and also the condition (price =
            // 20.00)

            UpdateItemSpec updateItemSpec = new UpdateItemSpec().withPrimaryKey("Id",
120)
                    .withReturnValues(ReturnValue.ALL_NEW).withUpdateExpression("set #p
= :val1")
                    .withConditionExpression("#p = :val2").withNameMap(new
NameMap().with("#p", "Price"))
                    .withValueMap(new ValueMap().withNumber(":val1",
25).withNumber(":val2", 20));

            UpdateItemOutcome outcome = table.updateItem(updateItemSpec);

            // Check the response.
            System.out.println("Printing item after conditional update to new
attribute...");
            System.out.println(outcome.getItem().toJSONPretty());
```

```
        } catch (Exception e) {
            System.err.println("Error updating item in " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void deleteItem() {

        Table table = dynamoDB.getTable(tableName);

        try {

            DeleteItemSpec deleteItemSpec = new DeleteItemSpec().withPrimaryKey("Id",
 120)
                    .withConditionExpression("#ip = :val").withNameMap(new
 NameMap().with("#ip", "InPublication"))
                    .withValueMap(new ValueMap().withBoolean(":val",
 false)).withReturnValues(ReturnValue.ALL_OLD);

            DeleteItemOutcome outcome = table.deleteItem(deleteItemSpec);

            // Check the response.
            System.out.println("Printing item that was deleted...");
            System.out.println(outcome.getItem().toJSONPretty());

        } catch (Exception e) {
            System.err.println("Error deleting item in " + tableName);
            System.err.println(e.getMessage());
        }
    }
}
```

Ejemplo: operaciones por lotes mediante la API de documentos de AWS SDK para Java

En esta sección se proporcionan ejemplos de operaciones de escritura y obtención por lote en Amazon DynamoDB mediante la API de documentos de AWS SDK para Java.

> **Note**
>
> Además, SDK para Java proporciona un modelo de persistencia de objetos, que le permite mapear las clases del lado del cliente a las tablas de DynamoDB. Este enfoque puede reducir la cantidad de código que hay que escribir. Para obtener más información, consulte Java 1.x: DynamoDBMapper.

Temas

- Ejemplo: operación de escritura por lotes mediante la API de documentos de AWS SDK para Java
- Ejemplo: operación de obtención por lotes mediante la API de documentos de AWS SDK para Java

Ejemplo: operación de escritura por lotes mediante la API de documentos de AWS SDK para Java

En el siguiente ejemplo de código Java se usa el método `batchWriteItem` para llevar a cabo las siguientes operaciones de colocación y eliminación:

- Colocar un elemento en la tabla `Forum`.
- Colocar un elemento y eliminar un elemento de la tabla `Thread`.

Al crear la solicitud de escritura por lotes, puede especificar cualquier cantidad de solicitudes de colocación y eliminación en una o varias tablas. Sin embargo, `batchWriteItem` limita el tamaño de una solicitud de escritura por lotes y el número de operaciones de colocación y eliminación que se pueden llevar a cabo en una misma operación de escritura por lotes. Si la solicitud supera estos límites, se rechaza la solicitud. Si la tabla no cuenta con suficiente desempeño provisionado para atender esta solicitud, los elementos de solicitud sin procesar se devuelven en la respuesta.

En el siguiente ejemplo se comprueba la respuesta para saber si contiene elementos de solicitud sin transformar. En caso afirmativo, entra en bucle y vuelve a enviar la solicitud `batchWriteItem` con elementos sin procesar. Si ha seguido los ejemplos de esta guía, ya debería haber creado las tablas `Forum` y `Thread`. También puede crear estas tablas y cargar los ejemplos de datos mediante programación. Para obtener más información, consulte Creación de ejemplos de tablas y carga de datos utilizando AWS SDK para Java.

Para obtener instrucciones paso a paso para probar el siguiente ejemplo, consulte Ejemplos de código Java.

Example

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.BatchWriteItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.TableWriteItems;
import com.amazonaws.services.dynamodbv2.model.WriteRequest;

public class DocumentAPIBatchWrite {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String forumTableName = "Forum";
    static String threadTableName = "Thread";

    public static void main(String[] args) throws IOException {

        writeMultipleItemsBatchWrite();

    }

    private static void writeMultipleItemsBatchWrite() {
        try {

            // Add a new item to Forum
            TableWriteItems forumTableWriteItems = new
 TableWriteItems(forumTableName) // Forum
                    .withItemsToPut(new Item().withPrimaryKey("Name", "Amazon
 RDS").withNumber("Threads", 0));

            // Add a new item, and delete an existing item, from Thread
            // This table has a partition key and range key, so need to specify
```

```
            // both of them
            TableWriteItems threadTableWriteItems = new
 TableWriteItems(threadTableName)
                    .withItemsToPut(
                            new Item().withPrimaryKey("ForumName", "Amazon RDS",
 "Subject", "Amazon RDS Thread 1")
                                    .withString("Message", "ElastiCache Thread 1
 message")
                                    .withStringSet("Tags", new
 HashSet<String>(Arrays.asList("cache", "in-memory"))))
                    .withHashAndRangeKeysToDelete("ForumName", "Subject", "Amazon S3",
 "S3 Thread 100");

            System.out.println("Making the request.");
            BatchWriteItemOutcome outcome =
 dynamoDB.batchWriteItem(forumTableWriteItems, threadTableWriteItems);

            do {

                // Check for unprocessed keys which could happen if you exceed
                // provisioned throughput

                Map<String, List<WriteRequest>> unprocessedItems =
 outcome.getUnprocessedItems();

                if (outcome.getUnprocessedItems().size() == 0) {
                    System.out.println("No unprocessed items found");
                } else {
                    System.out.println("Retrieving the unprocessed items");
                    outcome = dynamoDB.batchWriteItemUnprocessed(unprocessedItems);
                }

            } while (outcome.getUnprocessedItems().size() > 0);

        } catch (Exception e) {
            System.err.println("Failed to retrieve items: ");
            e.printStackTrace(System.err);
        }

    }

}
```

Ejemplo: operación de obtención por lotes mediante la API de documentos de AWS SDK para Java

En el siguiente ejemplo de código Java se usa el método `batchGetItem` para recuperar varios elementos de las tablas `Forum` y `Thread`. La solicitud `BatchGetItemRequest` especifica los nombres de las tablas y una lista de claves para cada elemento que se desea obtener. En el ejemplo se procesa la respuesta y se imprimen los elementos recuperados.

> **Note**
>
> En este ejemplo de código se supone que ya ha cargado datos en DynamoDB para su cuenta siguiendo las instrucciones de la sección Creación de tablas y carga de datos para ejemplos de código en DynamoDB.
> Para obtener instrucciones paso a paso acerca de cómo ejecutar el siguiente ejemplo, consulte Ejemplos de código Java.

Example

```
package com.amazonaws.codesamples.document;

import java.io.IOException;
import java.util.List;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.BatchGetItemOutcome;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.TableKeysAndAttributes;
import com.amazonaws.services.dynamodbv2.model.KeysAndAttributes;

public class DocumentAPIBatchGet {
    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String forumTableName = "Forum";
    static String threadTableName = "Thread";

    public static void main(String[] args) throws IOException {
        retrieveMultipleItemsBatchGet();
```

```
    }

    private static void retrieveMultipleItemsBatchGet() {

        try {

            TableKeysAndAttributes forumTableKeysAndAttributes = new
TableKeysAndAttributes(forumTableName);
            // Add a partition key
            forumTableKeysAndAttributes.addHashOnlyPrimaryKeys("Name", "Amazon S3",
"Amazon DynamoDB");

            TableKeysAndAttributes threadTableKeysAndAttributes = new
TableKeysAndAttributes(threadTableName);
            // Add a partition key and a sort key
            threadTableKeysAndAttributes.addHashAndRangePrimaryKeys("ForumName",
"Subject", "Amazon DynamoDB",
                    "DynamoDB Thread 1", "Amazon DynamoDB", "DynamoDB Thread 2",
"Amazon S3", "S3 Thread 1");

            System.out.println("Making the request.");

            BatchGetItemOutcome outcome =
dynamoDB.batchGetItem(forumTableKeysAndAttributes,
                    threadTableKeysAndAttributes);

            Map<String, KeysAndAttributes> unprocessed = null;

            do {
                for (String tableName : outcome.getTableItems().keySet()) {
                    System.out.println("Items in table " + tableName);
                    List<Item> items = outcome.getTableItems().get(tableName);
                    for (Item item : items) {
                        System.out.println(item.toJSONPretty());
                    }
                }

                // Check for unprocessed keys which could happen if you exceed
                // provisioned
                // throughput or reach the limit on response size.
                unprocessed = outcome.getUnprocessedKeys();

                if (unprocessed.isEmpty()) {
                    System.out.println("No unprocessed keys found");
```

```
                } else {
                    System.out.println("Retrieving the unprocessed keys");
                    outcome = dynamoDB.batchGetItemUnprocessed(unprocessed);
                }

            } while (!unprocessed.isEmpty());

        } catch (Exception e) {
            System.err.println("Failed to retrieve items.");
            System.err.println(e.getMessage());
        }

    }

}
```

Ejemplo: control de atributos de tipo binario mediante la API de documentos de AWS SDK para Java

En el siguiente ejemplo se ilustra cómo se controlan los atributos de tipo Binary. Además, se agrega un elemento a la tabla `Reply`. El elemento incluye un atributo de tipo Binary (`ExtendedMessage`) que almacena datos comprimidos. A continuación, en el ejemplo se recupera el elemento y se imprimen todos los valores de los atributos. Con fines ilustrativos, el ejemplo usa la clase `GZIPOutputStream` para comprimir un ejemplo de secuencia y asignársela al atributo `ExtendedMessage`. Cuando se recupera el atributo binario, se descomprime mediante la clase `GZIPInputStream`.

> **Note**
>
> Además, SDK para Java proporciona un modelo de persistencia de objetos, que le permite mapear las clases del lado del cliente a las tablas de DynamoDB. Este enfoque puede reducir la cantidad de código que hay que escribir. Para obtener más información, consulte Java 1.x: DynamoDBMapper.

Si ha seguido la sección Creación de tablas y carga de datos para ejemplos de código en DynamoDB, seguramente habrá creado ya la tabla `Reply`. También puede crear esta tabla mediante programación. Para obtener más información, consulte Creación de ejemplos de tablas y carga de datos utilizando AWS SDK para Java.

Para obtener instrucciones paso a paso para probar el siguiente ejemplo, consulte Ejemplos de código Java.

Example

```
package com.amazonaws.codesamples.document;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;
import java.util.zip.GZIPInputStream;
import java.util.zip.GZIPOutputStream;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.GetItemSpec;

public class DocumentAPIItemBinaryExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static String tableName = "Reply";
    static SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");

    public static void main(String[] args) throws IOException {
        try {

            // Format the primary key values
            String threadId = "Amazon DynamoDB#DynamoDB Thread 2";

            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));
            String replyDateTime = dateFormatter.format(new Date());

            // Add a new reply with a binary attribute type
```

```
            createItem(threadId, replyDateTime);

            // Retrieve the reply with a binary attribute type
            retrieveItem(threadId, replyDateTime);

            // clean up by deleting the item
            deleteItem(threadId, replyDateTime);
        } catch (Exception e) {
            System.err.println("Error running the binary attribute type example: " +
e);
            e.printStackTrace(System.err);
        }
    }

    public static void createItem(String threadId, String replyDateTime) throws
IOException {

        Table table = dynamoDB.getTable(tableName);

        // Craft a long message
        String messageInput = "Long message to be compressed in a lengthy forum reply";

        // Compress the long message
        ByteBuffer compressedMessage = compressString(messageInput.toString());

        table.putItem(new Item().withPrimaryKey("Id",
threadId).withString("ReplyDateTime", replyDateTime)
                .withString("Message", "Long message
follows").withBinary("ExtendedMessage", compressedMessage)
                .withString("PostedBy", "User A"));
    }

    public static void retrieveItem(String threadId, String replyDateTime) throws
IOException {

        Table table = dynamoDB.getTable(tableName);

        GetItemSpec spec = new GetItemSpec().withPrimaryKey("Id", threadId,
"ReplyDateTime", replyDateTime)
                .withConsistentRead(true);

        Item item = table.getItem(spec);

        // Uncompress the reply message and print
```

```
        String uncompressed =
uncompressString(ByteBuffer.wrap(item.getBinary("ExtendedMessage")));

        System.out.println("Reply message:\n" + " Id: " + item.getString("Id") + "\n" +
" ReplyDateTime: "
                + item.getString("ReplyDateTime") + "\n" + " PostedBy: " +
item.getString("PostedBy") + "\n"
                + " Message: "
                + item.getString("Message") + "\n" + " ExtendedMessage (uncompressed):
" + uncompressed + "\n");
    }

    public static void deleteItem(String threadId, String replyDateTime) {

        Table table = dynamoDB.getTable(tableName);
        table.deleteItem("Id", threadId, "ReplyDateTime", replyDateTime);
    }

    private static ByteBuffer compressString(String input) throws IOException {
        // Compress the UTF-8 encoded String into a byte[]
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        GZIPOutputStream os = new GZIPOutputStream(baos);
        os.write(input.getBytes("UTF-8"));
        os.close();
        baos.close();
        byte[] compressedBytes = baos.toByteArray();

        // The following code writes the compressed bytes to a ByteBuffer.
        // A simpler way to do this is by simply calling
        // ByteBuffer.wrap(compressedBytes);
        // However, the longer form below shows the importance of resetting the
        // position of the buffer
        // back to the beginning of the buffer if you are writing bytes directly
        // to it, since the SDK
        // will consider only the bytes after the current position when sending
        // data to DynamoDB.
        // Using the "wrap" method automatically resets the position to zero.
        ByteBuffer buffer = ByteBuffer.allocate(compressedBytes.length);
        buffer.put(compressedBytes, 0, compressedBytes.length);
        buffer.position(0); // Important: reset the position of the ByteBuffer
                            // to the beginning
        return buffer;
    }
```

```
    private static String uncompressString(ByteBuffer input) throws IOException {
        byte[] bytes = input.array();
        ByteArrayInputStream bais = new ByteArrayInputStream(bytes);
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        GZIPInputStream is = new GZIPInputStream(bais);

        int chunkSize = 1024;
        byte[] buffer = new byte[chunkSize];
        int length = 0;
        while ((length = is.read(buffer, 0, chunkSize)) != -1) {
            baos.write(buffer, 0, length);
        }

        String result = new String(baos.toByteArray(), "UTF-8");

        is.close();
        baos.close();
        bais.close();

        return result;
    }
}
```

Uso de elementos: .NET

Puede usar la API de bajo nivel AWS SDK para .NET para realizar operaciones típicas de creación, lectura, actualización y eliminación (CRUD, por sus siglas en inglés) en los elementos de una tabla. A continuación se indican los pasos que suelen llevarse a cabo para realizar operaciones CRUD a los datos mediante la API de bajo nivel de .NET:

1. Cree una instancia de la clase `AmazonDynamoDBClient` (el cliente).
2. Proporcione los parámetros requeridos específicos de la operación en el objeto de solicitud correspondiente.

   Por ejemplo, use el objeto de solicitud `PutItemRequest` para cargar un elemento y el objeto de solicitud `GetItemRequest` para recuperar un elemento existente.

   Puede usar el objeto de solicitud para proporcionar los parámetros necesarios y opcionales.
3. Ejecute el método apropiado proporcionado por el cliente pasándolo en el objeto de solicitud que ha creado en el paso anterior.

El cliente `AmazonDynamoDBClient` proporciona los métodos `PutItem`, `GetItem`, `UpdateItem` y `DeleteItem` para las operaciones CRUD.

Temas

- Colocación de un elemento
- Obtención de un elemento
- Actualización de un elemento
- Contador atómico
- Eliminación de un elemento
- Escritura por lotes: colocación y eliminación de varios elementos
- Obtención por lotes: obtención de varios elementos
- Ejemplo: operaciones CRUD con la API de bajo nivel de AWS SDK para .NET
- Ejemplo: operaciones por lotes con la API de bajo nivel de AWS SDK para .NET
- Ejemplo: control de atributos de tipo binario mediante la API de bajo nivel de AWS SDK para .NET

Colocación de un elemento

El método `PutItem` carga un elemento en una tabla. Si el elemento existe, sustituye el elemento completo.

> **Note**
>
> En lugar de ello, si prefiere actualizar solamente algunos atributos concretos, puede usar el método `UpdateItem`. Para obtener más información, consulte Actualización de un elemento.

A continuación se indican los pasos que hay que seguir para cargar un elemento mediante la API de bajo nivel del SDK para .NET:

1. Cree una instancia de la clase `AmazonDynamoDBClient`.
2. Cree una instancia de la clase `PutItemRequest` para proporcionar los parámetros requeridos.

   Para colocar un elemento, debe proporcionar el nombre de la tabla y el elemento.

3. Ejecute el método `PutItem` proporcionando el objeto `PutItemRequest` que creó en el paso anterior.

En el siguiente ejemplo de C# se ponen en práctica los pasos anteriores. En el ejemplo se carga un elemento en la tabla `ProductCatalog`.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new PutItemRequest
{
   TableName = tableName,
   Item = new Dictionary<string, AttributeValue>()
      {
          { "Id", new AttributeValue { N = "201" }},
          { "Title", new AttributeValue { S = "Book 201 Title" }},
          { "ISBN", new AttributeValue { S = "11-11-11-11" }},
          { "Price", new AttributeValue { S = "20.00" }},
          {
            "Authors",
            new AttributeValue
            { SS = new List<string>{"Author1", "Author2"}   }
          }
      }
};
client.PutItem(request);
```

En el ejemplo anterior, hemos cargado un elemento de libro que tiene los atributos `Id`, `Title`, `ISBN` y `Authors`. Tenga en cuenta `Id` es un atributo de tipo numérico y demás son de cadena. Autores es un conjunto `String`.

Especificación de parámetros opcionales

También puede usar el objeto `PutItemRequest` para proporcionar parámetros opcionales, como se muestra en el siguiente ejemplo de código C#. En el ejemplo se especifican los parámetros opcionales siguientes:

- `ExpressionAttributeNames`, `ExpressionAttributeValues` y `ConditionExpression` especifican que el elemento existente se puede sustituir únicamente si su atributo ISBN tiene un valor específico.
- El parámetro `ReturnValues` para solicitar el elemento anterior en la respuesta.

Example

```
var request = new PutItemRequest
 {
   TableName = tableName,
   Item = new Dictionary<string, AttributeValue>()
               {
                   { "Id", new AttributeValue { N = "104" }},
                   { "Title", new AttributeValue { S = "Book 104  Title" }},
                   { "ISBN", new AttributeValue { S = "444-4444444444" }},
                   { "Authors",
                     new AttributeValue { SS = new List<string>{"Author3"}}}
               },
    // Optional parameters.
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#I", "ISBN"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":isbn",new AttributeValue {S = "444-4444444444"}}
    },
    ConditionExpression = "#I = :isbn"

};
var response = client.PutItem(request);
```

Para obtener más información, consulte PutItem.

Obtención de un elemento

El método `GetItem` recupera un elemento.

> **Note**
>
> Para recuperar varios elementos, puede usar el método `BatchGetItem`. Para obtener más información, consulte Obtención por lotes: obtención de varios elementos.

A continuación se indican los pasos que hay que seguir para recuperar un elemento existente mediante la API de bajo nivel de AWS SDK para .NET.

1. Cree una instancia de la clase `AmazonDynamoDBClient`.
2. Cree una instancia de la clase `GetItemRequest` para proporcionar los parámetros requeridos.

   Para obtener un elemento, debe proporcionar el nombre de la tabla y la clave principal del elemento.
3. Ejecute el método `GetItem` proporcionando el objeto `GetItemRequest` que creó en el paso anterior.

En el siguiente ejemplo de C# se ponen en práctica los pasos anteriores. En el siguiente ejemplo se recupera un elemento de la tabla `ProductCatalog`.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new GetItemRequest
 {
   TableName = tableName,
   Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
 };
 var response = client.GetItem(request);

// Check the response.
var result = response.GetItemResult;
var attributeMap = result.Item; // Attribute list in the response.
```

Especificación de parámetros opcionales

También puede usar el objeto `GetItemRequest` para proporcionar parámetros opcionales, como se muestra en el siguiente ejemplo de código C#. En el ejemplo se especifican los parámetros opcionales siguientes:

- El parámetro `ProjectionExpression` para especificar los atributos que se van a recuperar.
- El parámetro `ConsistentRead` para realizar una lectura de consistencia alta. Para obtener más información sobre la consistencia de lectura, consulte Coherencia de lectura de DynamoDB.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new GetItemRequest
 {
   TableName = tableName,
   Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
   // Optional parameters.
   ProjectionExpression = "Id, ISBN, Title, Authors",
   ConsistentRead = true
 };

 var response = client.GetItem(request);

// Check the response.
var result = response.GetItemResult;
var attributeMap = result.Item;
```

Para obtener más información, consulte GetItem.

Actualización de un elemento

El método `UpdateItem` actualiza un elemento si existe. Puede usar la operación `UpdateItem` para actualizar los valores de atributos presentes, agregar atributos nuevos o eliminarlos de la colección existente. Si el elemento que tiene clave principal especificada no se encuentra, se agrega un nuevo elemento.

La operación `UpdateItem` se rige por las directrices siguientes:

- Si el elemento no existe, `UpdateItem` agrega un elemento nuevo utilizando la clave principal especificada en la información de entrada.
- Si el elemento existe, `UpdateItem` aplica las actualizaciones como se indica a continuación:
  - Sustituye los valores de los atributos existentes por los valores de la actualización.
  - Si el atributo que se proporciona en la información de entrada no existe, agrega un nuevo atributo al elemento.
  - Si el atributo de entrada es null, elimina el atributo, en caso de que esté presente.
  - Si se utiliza `ADD` para `Action`, puede agregar valores a un conjunto existente (de tipo String Set o Number Set) o bien sumar (si se usa un número positivo) o restar (si se usa un número negativo) matemáticamente un valor del atributo numérico existente.

> **Note**
>
> La operación `PutItem` también puede llevar a cabo una actualización. Para obtener más información, consulte Colocación de un elemento. Por ejemplo, si llama a `PutItem` para cargar un elemento y la clave principal ya existe, la operación `PutItem` sustituye el elemento completo. Tenga en cuenta que, si hay atributos en el elemento existente que no se especifican en la información de entrada, la operación `PutItem` los eliminará. Sin embargo, `UpdateItem` solo actualiza los atributos de entrada especificados. Todos los demás atributos existentes de ese elemento permanecen inalterados.

A continuación se indican los pasos que hay que seguir para actualizar un elemento existente mediante la API de bajo nivel del SDK para .NET:

1. Cree una instancia de la clase `AmazonDynamoDBClient`.
2. Cree una instancia de la clase `UpdateItemRequest` para proporcionar los parámetros requeridos.

   Se trata del objeto de solicitud en el que se describen todas las actualizaciones, tales como agregar atributos o actualizar o eliminar atributos existentes. Para eliminar un atributo, especifique su nombre con el valor null.
3. Ejecute el método `UpdateItem` proporcionando el objeto `UpdateItemRequest` que creó en el paso anterior.

En el siguiente ejemplo de código C# se ponen en práctica los pasos anteriores. En el ejemplo se actualiza un elemento libro de la tabla `ProductCatalog`. Se agrega un nuevo autor a la colección `Authors` y se elimina el atributo `ISBN` existente. También se reduce el precio en una unidad.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#A", "Authors"},
        {"#P", "Price"},
        {"#NA", "NewAttribute"},
        {"#I", "ISBN"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":auth",new AttributeValue { SS = {"Author YY","Author ZZ"}}},
        {":p",new AttributeValue {N = "1"}},
        {":newattr",new AttributeValue {S = "someValue"}},
    },

    // This expression does the following:
    // 1) Adds two new authors to the list
    // 2) Reduces the price
    // 3) Adds a new attribute to the item
    // 4) Removes the ISBN attribute from the item
    UpdateExpression = "ADD #A :auth SET #P = #P - :p, #NA = :newattr REMOVE #I"
};
var response = client.UpdateItem(request);
```

Especificación de parámetros opcionales

También puede usar el objeto `UpdateItemRequest` para proporcionar parámetros opcionales, como se muestra en el siguiente ejemplo de código C#. En él se especifican los parámetros opcionales siguientes:

- `ExpressionAttributeValues` y `ConditionExpression` para especificar que el precio solo puede actualizarse si el precio actual es 20.00.
- El parámetro `ReturnValues` para solicitar el elemento actualizado en la respuesta.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
{
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "202" } } },

    // Update price only if the current price is 20.00.
    ExpressionAttributeNames = new Dictionary<string,string>()
    {
        {"#P", "Price"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":newprice",new AttributeValue {N = "22"}},
        {":currprice",new AttributeValue {N = "20"}}
    },
    UpdateExpression = "SET #P = :newprice",
    ConditionExpression = "#P = :currprice",
    TableName = tableName,
    ReturnValues = "ALL_NEW" // Return all the attributes of the updated item.
};

var response = client.UpdateItem(request);
```

Para obtener más información, consulte UpdateItem.

Contador atómico

Puede usar `updateItem` para implementar un contador atómico y aumentar o reducir el valor de un atributo existente sin interferir con las demás solicitudes de escritura. Para actualizar un contador atómico, use `updateItem` con un atributo de tipo `Number` en el parámetro `UpdateExpression` y use `ADD` como `Action`.

En el siguiente ejemplo se pone en práctica lo anterior y se incrementa el atributo `Quantity` en una unidad.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new UpdateItemRequest
{
    Key = new Dictionary<string, AttributeValue>() { { "Id", new AttributeValue { N =
 "121" } } },
    ExpressionAttributeNames = new Dictionary<string, string>()
    {
        {"#Q", "Quantity"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":incr",new AttributeValue {N = "1"}}
    },
    UpdateExpression = "SET #Q = #Q + :incr",
    TableName = tableName
};

var response = client.UpdateItem(request);
```

Eliminación de un elemento

El método `DeleteItem` elimina un elemento de una tabla.

A continuación se indican los pasos que hay que seguir para eliminar un elemento mediante el API de bajo nivel del SDK para .NET.

1. Cree una instancia de la clase `AmazonDynamoDBClient`.
2. Cree una instancia de la clase `DeleteItemRequest` para proporcionar los parámetros requeridos.

   Para eliminar un elemento, se requieren el nombre de la tabla y la clave principal del elemento.
3. Ejecute el método `DeleteItem` proporcionando el objeto `DeleteItemRequest` que creó en el paso anterior.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new DeleteItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "201" } } },
};

var response = client.DeleteItem(request);
```

Especificación de parámetros opcionales

También puede usar el objeto `DeleteItemRequest` para proporcionar parámetros opcionales, como se muestra en el siguiente ejemplo de código C#. En él se especifican los parámetros opcionales siguientes:

- `ExpressionAttributeValues` y `ConditionExpression` para especificar que el elemento de libro solo se puede eliminar si está descatalogado (el valor del atributo InPublication es false).
- El parámetro `ReturnValues` para solicitar el elemento eliminado en la respuesta.

Example

```
var request = new DeleteItemRequest
{
    TableName = tableName,
    Key = new Dictionary<string,AttributeValue>() { { "Id", new AttributeValue { N =
 "201" } } },

    // Optional parameters.
    ReturnValues = "ALL_OLD",
    ExpressionAttributeNames = new Dictionary<string, string>()
    {
        {"#IP", "InPublication"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":inpub",new AttributeValue {BOOL = false}}
```

```
    },
    ConditionExpression = "#IP = :inpub"
};

var response = client.DeleteItem(request);
```

Para obtener más información, consulte DeleteItem.

Escritura por lotes: colocación y eliminación de varios elementos

La escritura por lotes se refiere a colocar y eliminar varios elementos en un lote. El método `BatchWriteItem` permite colocar y eliminar varios elementos de una o varias tablas con una sola llamada. A continuación se indican los pasos que hay que seguir para recuperar varios elementos mediante el API de bajo nivel del SDK para .NET.

1. Cree una instancia de la clase `AmazonDynamoDBClient`.
2. Cree una instancia de la clase `BatchWriteItemRequest` para describir todas las operaciones de colocación y eliminación.
3. Ejecute el método `BatchWriteItem` proporcionando el objeto `BatchWriteItemRequest` que creó en el paso anterior.
4. Procese la respuesta. Debe comprobar si en la respuesta se ha devuelto algún elemento de solicitud sin procesar. Esto puede ocurrir si se alcanza la cuota de rendimiento aprovisionado o se produce algún otro error transitorio. Además, DynamoDB limita el tamaño de la solicitud y el número de operaciones que se pueden especificar en ella. Si supera estos límites, DynamoDB rechaza la solicitud. Para obtener más información, consulte BatchWriteItem.

En el siguiente ejemplo de código C# se ponen en práctica los pasos anteriores. En el siguiente ejemplo se crea una solicitud `BatchWriteItemRequest` para realizar las siguientes operaciones de escritura:

- Colocar un elemento en la tabla `Forum`.
- Colocar y eliminar un elemento en la tabla `Thread`

A continuación, el código ejecuta `BatchWriteItem` para llevar a cabo una operación por lote.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

string table1Name = "Forum";
```

```
string table2Name = "Thread";

var request = new BatchWriteItemRequest
 {
   RequestItems = new Dictionary<string, List<WriteRequest>>
    {
      {
        table1Name, new List<WriteRequest>
        {
          new WriteRequest
          {
             PutRequest = new PutRequest
             {
                Item = new Dictionary<string,AttributeValue>
                {
                  { "Name", new AttributeValue { S = "Amazon S3 forum" } },
                  { "Threads", new AttributeValue { N = "0" }}
                }
             }
          }
        }
      } ,
      {
        table2Name, new List<WriteRequest>
        {
          new WriteRequest
          {
            PutRequest = new PutRequest
            {
               Item = new Dictionary<string,AttributeValue>
               {
                 { "ForumName", new AttributeValue { S = "Amazon S3 forum" } },
                 { "Subject", new AttributeValue { S = "My sample question" } },
                 { "Message", new AttributeValue { S = "Message Text." } },
                 { "KeywordTags", new AttributeValue { SS = new List<string> { "Amazon
 S3", "Bucket" }  } }
               }
            }
          },
          new WriteRequest
          {
             DeleteRequest = new DeleteRequest
             {
                Key = new Dictionary<string,AttributeValue>()
```

```
                {
                   { "ForumName", new AttributeValue { S = "Some forum name" } },
                   { "Subject", new AttributeValue { S = "Some subject" } }
                }
             }
          }
        }
      }
    }
 };
response = client.BatchWriteItem(request);
```

Para ver un ejemplo práctico, consulte Ejemplo: operaciones por lotes con la API de bajo nivel de AWS SDK para .NET.

Obtención por lotes: obtención de varios elementos

El método `BatchGetItem` permite recuperar varios elementos de una o varias tablas.

> **Note**
>
> Para recuperar un solo elemento, puede usar el método `GetItem`.

A continuación se indican los pasos que hay que seguir para recuperar varios elementos mediante la API de bajo nivel AWS SDK para .NET.

1. Cree una instancia de la clase `AmazonDynamoDBClient`.
2. Cree una instancia de la clase `BatchGetItemRequest` para proporcionar los parámetros requeridos.

   Para recuperar varios elementos, es obligatorio indicar el nombre de la tabla y una lista de valores de clave principal.
3. Ejecute el método `BatchGetItem` proporcionando el objeto `BatchGetItemRequest` que creó en el paso anterior.
4. Procese la respuesta. Debe comprobar si han quedado claves sin procesar, lo que podría ocurrir si se alcanza la cuota de rendimiento aprovisionada o se produce cualquier otro error transitorio.

En el siguiente ejemplo de código C# se ponen en práctica los pasos anteriores. En el ejemplo se recuperan elementos de dos tablas, `Forum` y `Thread`. La solicitud especifica dos elementos en la tabla `Forum` y tres en la tabla `Thread`. La respuesta incluye elementos de ambas tablas. En el código se muestra cómo procesar la respuesta.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

string table1Name = "Forum";
string table2Name = "Thread";

var request = new BatchGetItemRequest
{
  RequestItems = new Dictionary<string, KeysAndAttributes>()
  {
    { table1Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "DynamoDB" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "Amazon S3" } }
          }
        }
      }
    },
    {
      table2Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "ForumName", new AttributeValue { S = "DynamoDB" } },
            { "Subject", new AttributeValue { S = "DynamoDB Thread 1" } }
          },
          new Dictionary<string, AttributeValue>()
```

```
          {
            { "ForumName", new AttributeValue { S = "DynamoDB" } },
            { "Subject", new AttributeValue { S = "DynamoDB Thread 2" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "ForumName", new AttributeValue { S = "Amazon S3" } },
            { "Subject", new AttributeValue { S = "Amazon S3 Thread 1" } }
          }
        }
      }
    }
  }
};

var response = client.BatchGetItem(request);

// Check the response.
var result = response.BatchGetItemResult;
var responses = result.Responses; // The attribute list in the response.

var table1Results = responses[table1Name];
Console.WriteLine("Items in table {0}" + table1Name);
foreach (var item1 in table1Results.Items)
{
  PrintItem(item1);
}

var table2Results = responses[table2Name];
Console.WriteLine("Items in table {1}" + table2Name);
foreach (var item2 in table2Results.Items)
{
  PrintItem(item2);
}
// Any unprocessed keys? could happen if you exceed ProvisionedThroughput or some other
 error.
Dictionary<string, KeysAndAttributes> unprocessedKeys = result.UnprocessedKeys;
foreach (KeyValuePair<string, KeysAndAttributes> pair in unprocessedKeys)
{
    Console.WriteLine(pair.Key, pair.Value);
}
```

Especificación de parámetros opcionales

También puede usar el objeto `BatchGetItemRequest` para proporcionar parámetros opcionales, como se muestra en el siguiente ejemplo de código C#. En el siguiente ejemplo de código se recuperan dos elementos de la tabla `Forum`. En él se especifica el parámetro opcional siguiente:

- El parámetro `ProjectionExpression` para especificar los atributos que se van a recuperar.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

string table1Name = "Forum";

var request = new BatchGetItemRequest
{
  RequestItems = new Dictionary<string, KeysAndAttributes>()
  {
    { table1Name,
      new KeysAndAttributes
      {
        Keys = new List<Dictionary<string, AttributeValue>>()
        {
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "DynamoDB" } }
          },
          new Dictionary<string, AttributeValue>()
          {
            { "Name", new AttributeValue { S = "Amazon S3" } }
          }
        }
      },
      // Optional - name of an attribute to retrieve.
      ProjectionExpression = "Title"
    }
  }
};

var response = client.BatchGetItem(request);
```

Para obtener más información, consulte BatchGetItem.

Ejemplo: operaciones CRUD con la API de bajo nivel de AWS SDK para .NET

En el siguiente ejemplo de código C# se ilustran las operaciones de creación, lectura, actualización y eliminación (CRUD, Create, Read, Update and Delete) en un elemento de Amazon DynamoDB. En el ejemplo se agrega un elemento a la tabla `ProductCatalog`, se recupera, se llevan a cabo varias actualizaciones y, por último, se elimina. Si no ha creado esta tabla, también puede crearla mediante programación. Para obtener más información, consulte Creación de ejemplos de tablas y carga de datos utilizando AWS SDK para .NET.

Para obtener instrucciones paso a paso para probar el siguiente ejemplo, consulte Ejemplos de código .NET.

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelItemCRUDExample
    {
        private static string tableName = "ProductCatalog";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                CreateItem();
                RetrieveItem();

                // Perform various updates.
                UpdateMultipleAttributes();
                UpdateExistingAttributeConditionally();

                // Delete item.
                DeleteItem();
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
```

```
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
        }

        private static void CreateItem()
        {
            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  }},
                { "Title", new AttributeValue {
                      S = "Book 201 Title"
                  }},
                { "ISBN", new AttributeValue {
                      S = "11-11-11-11"
                  }},
                { "Authors", new AttributeValue {
                      SS = new List<string>{"Author1", "Author2" }
                  }},
                { "Price", new AttributeValue {
                      N = "20.00"
                  }},
                { "Dimensions", new AttributeValue {
                      S = "8.5x11.0x.75"
                  }},
                { "InPublication", new AttributeValue {
                      BOOL = false
                  } }
            }
            };
            client.PutItem(request);
        }

        private static void RetrieveItem()
        {
```

```
            var request = new GetItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },
                ProjectionExpression = "Id, ISBN, Title, Authors",
                ConsistentRead = true
            };
            var response = client.GetItem(request);

            // Check the response.
            var attributeList = response.Item; // attribute list in the response.
            Console.WriteLine("\nPrinting item after retrieving it ............");
            PrintItem(attributeList);
        }

        private static void UpdateMultipleAttributes()
        {
            var request = new UpdateItemRequest
            {
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },
                // Perform the following updates:
                // 1) Add two new authors to the list
                // 1) Set a new attribute
                // 2) Remove the ISBN attribute
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#A","Authors"},
                {"#NA","NewAttribute"},
                {"#I","ISBN"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":auth",new AttributeValue {
                     SS = {"Author YY", "Author ZZ"}
```

```
                 }},
                {":new",new AttributeValue {
                     S = "New Value"
                 }}
            },

                UpdateExpression = "ADD #A :auth SET #NA = :new REMOVE #I",

                TableName = tableName,
                ReturnValues = "ALL_NEW" // Give me all attributes of the updated item.
            };
            var response = client.UpdateItem(request);

            // Check the response.
            var attributeList = response.Attributes; // attribute list in the response.
                                                     // print attributeList.
            Console.WriteLine("\nPrinting item after multiple attribute
update .............");
            PrintItem(attributeList);
        }

        private static void UpdateExistingAttributeConditionally()
        {
            var request = new UpdateItemRequest
            {
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#P", "Price"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":newprice",new AttributeValue {
                     N = "22.00"
                 }},
                {":currprice",new AttributeValue {
                     N = "20.00"
                 }}
            },
```

```
                // This updates price only if current price is 20.00.
                UpdateExpression = "SET #P = :newprice",
                ConditionExpression = "#P = :currprice",

                TableName = tableName,
                ReturnValues = "ALL_NEW" // Give me all attributes of the updated item.
            };
            var response = client.UpdateItem(request);

            // Check the response.
            var attributeList = response.Attributes; // attribute list in the response.
            Console.WriteLine("\nPrinting item after updating price value
conditionally ............");
            PrintItem(attributeList);
        }

        private static void DeleteItem()
        {
            var request = new DeleteItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      N = "1000"
                  } }
            },

                // Return the entire item as it appeared before the update.
                ReturnValues = "ALL_OLD",
                ExpressionAttributeNames = new Dictionary<string, string>()
            {
                {"#IP", "InPublication"}
            },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
            {
                {":inpub",new AttributeValue {
                     BOOL = false
                 }}
            },
                ConditionExpression = "#IP = :inpub"
            };

            var response = client.DeleteItem(request);
```

```
            // Check the response.
            var attributeList = response.Attributes; // Attribute list in the response.
                                                     // Print item.
            Console.WriteLine("\nPrinting item that was just deleted ............");
            PrintItem(attributeList);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }
            Console.WriteLine("************************************************");
        }
    }
}
```

Ejemplo: operaciones por lotes con la API de bajo nivel de AWS SDK para .NET

Temas

- Ejemplo: operación de escritura por lote mediante la API de bajo nivel de AWS SDK para .NET
- Ejemplo: operación de obtención por lotes mediante la API de bajo nivel de AWS SDK para .NET

En esta sección se proporcionan ejemplos de las operaciones de escritura por lote y obtención por lote que Amazon DynamoDB admite.

Ejemplo: operación de escritura por lote mediante la API de bajo nivel de AWS SDK para .NET

En el siguiente ejemplo de código C# se usa el método `BatchWriteItem` para llevar a cabo las siguientes operaciones de colocación y eliminación:

- Colocar un elemento en la tabla `Forum`.
- Colocar un elemento y eliminar un elemento de la tabla `Thread`.

Al crear la solicitud de escritura por lotes, puede especificar cualquier cantidad de solicitudes de colocación y eliminación en una o varias tablas. Sin embargo, `BatchWriteItem` de DynamoDB limita el tamaño de una solicitud de escritura por lote y el número de operaciones de colocación y eliminación que se pueden llevar a cabo en una misma operación de escritura por lote. Para obtener más información, consulte BatchWriteItem. Si la solicitud supera estos límites, se rechaza la solicitud. Si la tabla no cuenta con suficiente desempeño provisionado para atender esta solicitud, los elementos de solicitud sin procesar se devuelven en la respuesta.

En el siguiente ejemplo se comprueba la respuesta para saber si contiene elementos de solicitud sin transformar. En caso afirmativo, entra en bucle y vuelve a enviar la solicitud `BatchWriteItem` con elementos sin procesar. También puede crear estos ejemplos de tablas y cargar los ejemplos de datos mediante programación. Para obtener más información, consulte Creación de ejemplos de tablas y carga de datos utilizando AWS SDK para .NET.

Para obtener instrucciones paso a paso para probar el siguiente ejemplo, consulte Ejemplos de código .NET.

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelBatchWrite
    {
        private static string table1Name = "Forum";
        private static string table2Name = "Thread";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
```

```
        static void Main(string[] args)
        {
            try
            {
                TestBatchWrite();
            }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }

            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }

        private static void TestBatchWrite()
        {
            var request = new BatchWriteItemRequest
            {
                ReturnConsumedCapacity = "TOTAL",
                RequestItems = new Dictionary<string, List<WriteRequest>>
            {
                {
                    table1Name, new List<WriteRequest>
                    {
                        new WriteRequest
                        {
                            PutRequest = new PutRequest
                            {
                                Item = new Dictionary<string, AttributeValue>
                                {
                                    { "Name", new AttributeValue {
                                          S = "S3 forum"
                                      } },
                                    { "Threads", new AttributeValue {
                                          N = "0"
                                      }}
                                }
                            }
                        }
                    }
                },
                {
                    table2Name, new List<WriteRequest>
                    {
                        new WriteRequest
```

```
                        {
                            PutRequest = new PutRequest
                            {
                                Item = new Dictionary<string, AttributeValue>
                                {
                                    { "ForumName", new AttributeValue {
                                          S = "S3 forum"
                                      } },
                                    { "Subject", new AttributeValue {
                                          S = "My sample question"
                                      } },
                                    { "Message", new AttributeValue {
                                          S = "Message Text."
                                      } },
                                    { "KeywordTags", new AttributeValue {
                                          SS = new List<string> { "S3", "Bucket" }
                                      } }
                                }
                            }
                        },
                        new WriteRequest
                        {
                            // For the operation to delete an item, if you provide a
primary key value
                            // that does not exist in the table, there is no error, it
is just a no-op.
                            DeleteRequest = new DeleteRequest
                            {
                                Key = new Dictionary<string, AttributeValue>()
                                {
                                    { "ForumName",  new AttributeValue {
                                          S = "Some partition key value"
                                      } },
                                    { "Subject", new AttributeValue {
                                          S = "Some sort key value"
                                      } }
                                }
                            }
                        }
                    }
                }
            }
            };
```

```
            CallBatchWriteTillCompletion(request);
        }

        private static void CallBatchWriteTillCompletion(BatchWriteItemRequest request)
        {
            BatchWriteItemResponse response;

            int callCount = 0;
            do
            {
                Console.WriteLine("Making request");
                response = client.BatchWriteItem(request);
                callCount++;

                // Check the response.

                var tableConsumedCapacities = response.ConsumedCapacity;
                var unprocessed = response.UnprocessedItems;

                Console.WriteLine("Per-table consumed capacity");
                foreach (var tableConsumedCapacity in tableConsumedCapacities)
                {
                    Console.WriteLine("{0} - {1}", tableConsumedCapacity.TableName,
 tableConsumedCapacity.CapacityUnits);
                }

                Console.WriteLine("Unprocessed");
                foreach (var unp in unprocessed)
                {
                    Console.WriteLine("{0} - {1}", unp.Key, unp.Value.Count);
                }
                Console.WriteLine();

                // For the next iteration, the request will have unprocessed items.
                request.RequestItems = unprocessed;
            } while (response.UnprocessedItems.Count > 0);

            Console.WriteLine("Total # of batch write API calls made: {0}", callCount);
        }
    }
}
```

Ejemplo: operación de obtención por lotes mediante la API de bajo nivel de AWS SDK para .NET

En el siguiente ejemplo de código C# se usa el método `BatchGetItem` para recuperar varios elementos de las tablas `Forum` y `Thread` en Amazon DynamoDB. La solicitud `BatchGetItemRequest` especifica los nombres de las tablas y una lista de claves principales para cada tabla. En el ejemplo se procesa la respuesta y se imprimen los elementos recuperados.

Para obtener instrucciones paso a paso para probar el siguiente ejemplo, consulte Ejemplos de código .NET.

Example

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelBatchGet
    {
        private static string table1Name = "Forum";
        private static string table2Name = "Thread";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
            {
                RetrieveMultipleItemsBatchGet();

                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void RetrieveMultipleItemsBatchGet()
        {
            var request = new BatchGetItemRequest
            {
```

```
            RequestItems = new Dictionary<string, KeysAndAttributes>()
        {
            { table1Name,
              new KeysAndAttributes
              {
                  Keys = new List<Dictionary<string, AttributeValue> >()
                  {
                      new Dictionary<string, AttributeValue>()
                      {
                          { "Name", new AttributeValue {
                            S = "Amazon DynamoDB"
                      } }
                      },
                      new Dictionary<string, AttributeValue>()
                      {
                          { "Name", new AttributeValue {
                            S = "Amazon S3"
                      } }
                      }
                  }
              }},
            {
                table2Name,
                new KeysAndAttributes
                {
                    Keys = new List<Dictionary<string, AttributeValue> >()
                    {
                        new Dictionary<string, AttributeValue>()
                        {
                            { "ForumName", new AttributeValue {
                                  S = "Amazon DynamoDB"
                              } },
                            { "Subject", new AttributeValue {
                                  S = "DynamoDB Thread 1"
                              } }
                        },
                        new Dictionary<string, AttributeValue>()
                        {
                            { "ForumName", new AttributeValue {
                                  S = "Amazon DynamoDB"
                              } },
                            { "Subject", new AttributeValue {
                                  S = "DynamoDB Thread 2"
                              } }
```

```
                        },
                        new Dictionary<string, AttributeValue>()
                        {
                            { "ForumName", new AttributeValue {
                                  S = "Amazon S3"
                              } },
                            { "Subject", new AttributeValue {
                                  S = "S3 Thread 1"
                              } }
                        }
                    }
                }
            }
            };

            BatchGetItemResponse response;
            do
            {
                Console.WriteLine("Making request");
                response = client.BatchGetItem(request);

                // Check the response.
                var responses = response.Responses; // Attribute list in the response.

                foreach (var tableResponse in responses)
                {
                    var tableResults = tableResponse.Value;
                    Console.WriteLine("Items retrieved from table {0}",
tableResponse.Key);
                    foreach (var item1 in tableResults)
                    {
                        PrintItem(item1);
                    }
                }

                // Any unprocessed keys? could happen if you exceed
ProvisionedThroughput or some other error.
                Dictionary<string, KeysAndAttributes> unprocessedKeys =
response.UnprocessedKeys;
                foreach (var unprocessedTableKeys in unprocessedKeys)
                {
                    // Print table name.
                    Console.WriteLine(unprocessedTableKeys.Key);
```

```
                    // Print unprocessed primary keys.
                    foreach (var key in unprocessedTableKeys.Value.Keys)
                    {
                        PrintItem(key);
                    }
                }

                request.RequestItems = unprocessedKeys;
            } while (response.UnprocessedKeys.Count > 0);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
 value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }
            Console.WriteLine("************************************************");
        }
    }
}
```

Ejemplo: control de atributos de tipo binario mediante la API de bajo nivel de AWS SDK para .NET

En el siguiente ejemplo de código C# se ilustra cómo se controlan los atributos de tipo Binary. Además, se agrega un elemento a la tabla `Reply`. El elemento incluye un atributo de tipo Binary (`ExtendedMessage`) que almacena datos comprimidos. A continuación, en el ejemplo se recupera el elemento y se imprimen todos los valores de los atributos. Con fines ilustrativos, el ejemplo usa la clase `GZipStream` para comprimir un ejemplo de secuencia y asignársela al atributo `ExtendedMessage`; después, la descomprime al imprimir el valor del atributo.

Para obtener instrucciones paso a paso para probar el siguiente ejemplo, consulte Ejemplos de código .NET.

Example

```
using System;
using System.Collections.Generic;
using System.IO;
using System.IO.Compression;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace com.amazonaws.codesamples
{
    class LowLevelItemBinaryExample
    {
        private static string tableName = "Reply";
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            // Reply table primary key.
            string replyIdPartitionKey = "Amazon DynamoDB#DynamoDB Thread 1";
            string replyDateTimeSortKey = Convert.ToString(DateTime.UtcNow);

            try
            {
                CreateItem(replyIdPartitionKey, replyDateTimeSortKey);
                RetrieveItem(replyIdPartitionKey, replyDateTimeSortKey);
                // Delete item.
                DeleteItem(replyIdPartitionKey, replyDateTimeSortKey);
                Console.WriteLine("To continue, press Enter");
                Console.ReadLine();
            }
            catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void CreateItem(string partitionKey, string sortKey)
        {
            MemoryStream compressedMessage = ToGzipMemoryStream("Some long extended
 message to compress.");
```

```
            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      S = partitionKey
                  }},
                { "ReplyDateTime", new AttributeValue {
                      S = sortKey
                  }},
                { "Subject", new AttributeValue {
                      S = "Binary type "
                  }},
                { "Message", new AttributeValue {
                      S = "Some message about the binary type"
                  }},
                { "ExtendedMessage", new AttributeValue {
                      B = compressedMessage
                  }}
            }
            };
            client.PutItem(request);
        }

        private static void RetrieveItem(string partitionKey, string sortKey)
        {
            var request = new GetItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      S = partitionKey
                  } },
                { "ReplyDateTime", new AttributeValue {
                      S = sortKey
                  } }
            },
                ConsistentRead = true
            };
            var response = client.GetItem(request);

            // Check the response.
```

```
            var attributeList = response.Item; // attribute list in the response.
            Console.WriteLine("\nPrinting item after retrieving it ............");

            PrintItem(attributeList);
        }

        private static void DeleteItem(string partitionKey, string sortKey)
        {
            var request = new DeleteItemRequest
            {
                TableName = tableName,
                Key = new Dictionary<string, AttributeValue>()
            {
                { "Id", new AttributeValue {
                      S = partitionKey
                  } },
                { "ReplyDateTime", new AttributeValue {
                      S = sortKey
                  } }
            }
            };
            var response = client.DeleteItem(request);
        }

        private static void PrintItem(Dictionary<string, AttributeValue> attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
value.SS.ToArray()) + "]") +
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
value.NS.ToArray()) + "]") +
                    (value.B == null ? "" : "B=[" + FromGzipMemoryStream(value.B) +
"]")
                    );
            }
            Console.WriteLine("*****************************************************");
```

```
        }

        private static MemoryStream ToGzipMemoryStream(string value)
        {
            MemoryStream output = new MemoryStream();
            using (GZipStream zipStream = new GZipStream(output,
 CompressionMode.Compress, true))
            using (StreamWriter writer = new StreamWriter(zipStream))
            {
                writer.Write(value);
            }
            return output;
        }

        private static string FromGzipMemoryStream(MemoryStream stream)
        {
            using (GZipStream zipStream = new GZipStream(stream,
 CompressionMode.Decompress))
            using (StreamReader reader = new StreamReader(zipStream))
            {
                return reader.ReadToEnd();
            }
        }
    }
}
```

Mejora del acceso con índices secundarios en DynamoDB

Amazon DynamoDB proporciona un acceso rápido a los elementos de una tabla especificando sus valores de clave principal. Sin embargo, muchas aplicaciones podrían beneficiarse de disponer de una o varias claves secundarias (o alternativas) para permitir un acceso eficiente a datos con otros atributos aparte de la clave principal. Para responder a esta necesidad, puede crear uno o varios índices secundarios en una tabla y emitir solicitudes `Query` o `Scan` para estos índices.

Un índice secundario es una estructura de datos que contiene un subconjunto de atributos de una tabla, además de una clave alternativa para admitir las operaciones `Query`. Puede recuperar datos del índice usando una operación `Query` prácticamente de la misma forma que `Query` se usa en una tabla. Una tabla puede tener varios índices secundarios, lo que permite a las aplicaciones obtener acceso a distintos patrones de consulta.

> **Note**
>
> También se pueden utilizar operaciones `Scan` en los índices, de un modo bastante similar a como `Scan` se usaría en una tabla.
> Las políticas basadas en recursos no admiten actualmente el acceso entre cuentas para operaciones de análisis de índices secundarios.

Cada índice secundario está asociado exactamente con una tabla de la que obtiene sus datos. Esta se denomina tabla base del índice. Al crear un índice, se define una clave alternativa para él (clave de partición y clave de ordenación). También se definen los atributos de la tabla base que se desea proyectar, o copiar, en el índice. DynamoDB copia estos atributos en el índice, junto con los atributos de clave principal de la tabla base. A partir de ese momento, puede consultar o examinar el índice exactamente igual que una tabla.

DynamoDB mantiene automáticamente cada índice secundario. Cuando se agregan, modifican o eliminan elementos en la tabla base, los índices basados en esa tabla se actualizan también para reflejar estos cambios.

DynamoDB admite dos tipos de índices secundarios:

- Índice secundario global: índice con una clave de partición y una clave de ordenación que pueden diferir de las claves de la tabla base. Un índice secundario global se considera "global" porque las consultas que se realizan en el índice pueden abarcar todos los datos de la tabla base y todas las particiones. Un índice secundario global se almacena en su propio espacio de partición lejos de la tabla base y se escala por separado de la tabla base.
- Índice secundario local: índice que tiene la misma clave de partición que la tabla base, pero una clave de ordenación distinta. Un índice secundario local se considera "local" en el sentido de que el ámbito de todas sus particiones se corresponde con una partición de la tabla base que tiene el mismo valor de clave de partición.

Para ver una comparación de los índices secundarios globales y los índices secundarios locales, consulte este vídeo.

Tomar la decisión correcta entre GSI y LSI

Temas

- Uso de índices secundarios globales en DynamoDB

- Índices secundarios locales

Debe tener en cuenta los requisitos de la aplicación al determinar qué tipo de índice va a utilizar. En la siguiente tabla se muestran las principales diferencias entre un índice secundario global y un índice secundario local.

Característica	Índice secundario global	Índice secundario local
Esquema de claves	La clave principal de un índice secundario global puede ser simple (clave de partición) o compuesta (clave de partición y clave de ordenación).	La clave principal de un índice secundario local debe ser compuesta (clave de partición y clave de ordenación).
Atributos de clave	La clave de partición y la clave de ordenación (si procede) del índice pueden ser cualesqui era atributos de tipo String, Number o Binary de la tabla base.	La clave de partición del índice es el mismo atributo que la clave de partición de la tabla base. La clave de ordenación puede ser cualquier atributo de tipo String, Number o Binary de la tabla base.
Restricciones de tamaño por valor de clave de partición	No hay restricciones de tamaño para los índices secundarios globales.	Para cada valor de clave de partición, el tamaño total de todos los elementos indexados debe ser de 10 GB o menos.
Operaciones online con índices	Los índices secundarios globales se pueden crear al mismo tiempo que se crea una tabla. También puede agregar un nuevo índice secundario global a una tabla existente o eliminar un índice secundario global existente.	Los índices secundarios locales se crean a la vez que se crea la tabla. No se puede agregar un índice secundari o local a una tabla existente, ni tampoco se puede eliminar ningún índice secundario local que ya exista.

Característica	Índice secundario global	Índice secundario local
	Para obtener más información, consulte Administración de índices secundarios globales en DynamoDB.	
Consultas y particiones	Un índice secundario global permite realizar consultas en toda la tabla y en todas las particiones.	Un índice secundario local permite consultar una sola partición, según lo especificado por el valor de clave de partición de la consulta.
Consistencia de lectura	Las consultas a los índices secundarios globales solo admiten la consistencia final.	Cuando se realiza una consulta en un índice secundario local, se puede elegir entre la consistencia final o alta.
Consumo de desempeño provisionado	Cada índice secundario global tiene su propia configuración de rendimiento aprovisionado para la actividad de lectura y escritura. Las consultas o exámenes de un índice secundario global consumen unidades de capacidad del índice, no de la tabla base. Esto mismo sucede con las actualizaciones de los índices secundarios globales debidas a escrituras en la tabla. Un índice secundario global asociado a las tablas globales consume unidades de capacidad de escritura.	Las consultas o análisis de un índice secundario local consumen unidades de capacidad de lectura de la tabla base. Al escribir en una tabla, sus índices secundarios locales también se actualizan y estas actualizaciones consumen unidades de capacidad de escritura de la tabla base. Un índice secundario local asociado a las tablas globales consume unidades de capacidad de escritura replicadas.

Característica	Índice secundario global	Índice secundario local
Atributos proyectados	Cuando se consulta o analiza un índice secundario global, solo se pueden solicitar los atributos que se han proyectado en él. DynamoDB no recupera ningún atributo de la tabla.	Cuando se consulta o analiza un índice secundario local, se pueden solicitar los atributos que no se hayan proyectado en él. DynamoDB recuperará automáticamente estos atributos de la tabla.

Si desea crear más de una tabla con índices secundarios, debe hacerlo de forma secuencial. Por ejemplo, tendría que crear la primera tabla y esperar a que su estado fuese `ACTIVE`, luego la siguiente tabla y esperar a que adquiriese el estado `ACTIVE`, y así sucesivamente. Si intenta crear simultáneamente más de una tabla con un índice secundario, DynamoDB devuelve una excepción `LimitExceededException`.

Cada índice secundario usa la misma clase de tabla y el mismo modo de capacidad que la tabla base a la que está asociado. Para cada índice secundario debe especificar lo siguiente:

- Tipo de índice que se va a crear, ya sea un índice secundario global o un índice secundario local.
- El nombre del índice. Las reglas de nomenclatura de los índices son las mismas que para las tablas, como se indica en Cuotas en Amazon DynamoDB. El nombre debe ser único para la tabla base al que está asociado, pero puede utilizar el mismo nombre para índices que estén asociados a tablas base distintas.
- El esquema de claves del índice. Cada atributo del esquema de claves del índice debe ser un atributo de nivel superior debe de tipo `String`, `Number` y `Binary`. No se permiten otros tipos de datos, como los documentos y los conjuntos. Los demás requisitos del esquema de claves dependen del tipo de índice:
  - Si se trata de un índice secundario global, la clave de partición puede ser cualquier atributo escalar de la tabla base. La clave de ordenación es opcional y también puede ser cualquier atributo escalar de la tabla base.
  - Si se trata de un índice secundario local, la clave de partición debe ser igual que la tabla de partición de la tabla base y la clave de ordenación debe ser un atributo sin clave de la tabla base.
- Los atributos adicionales, si los hay, de la tabla base que se proyectarán en el índice. Estos atributos se agregan a los atributos de clave de la tabla, que se proyectan de forma automática en

cada índice. Puede proyectar atributos de cualquier tipo de datos, incluidos escalares, documentos y conjuntos.

- Los ajustes de desempeño provisionado del índice, si es preciso:
  - Para un índice secundario global, debe especificar los ajustes de unidades de capacidad de lectura y escritura. Estos ajustes de desempeño provisionado son independientes de los ajustes de la tabla base.
  - Para un índice secundario local, no es preciso especificar los ajustes de unidades de capacidad de lectura y escritura. Todas las operaciones de lectura y escritura en un índice secundario local consumen el rendimiento aprovisionado configurado para su tabla base.

Para disfrutar de la máxima flexibilidad en las consultas, puede crear hasta 20 índices secundarios globales (cuota predeterminada) y hasta 5 índices secundarios locales por tabla.

Para obtener un listado detallado de índices secundarios en una tabla, utilice la operación `DescribeTable`. `DescribeTable` devuelve el nombre, el tamaño de almacenamiento y el recuento de elementos de cada índice secundario de la tabla. Estos valores no se actualizan en tiempo real, sino aproximadamente cada seis horas.

Para obtener acceso a los datos de un índice secundario, utilice las operaciones `Query` o `Scan`. Debe especificar el nombre de la tabla base y el nombre del índice que se desea utilizar, los atributos que se devolverán en los resultados y las expresiones de condición o filtros que se van a aplicar. DynamoDB puede devolver los resultados en orden ascendente o descendente.

Al eliminar una tabla, todos los índices asociados a ella se eliminan también.

Para ver las prácticas recomendadas, consulte Prácticas recomendadas para utilizar índices secundarios en DynamoDB.

Uso de índices secundarios globales en DynamoDB

Algunas aplicaciones pueden necesitar llevar a cabo muy diversos tipos de consultas en las que se usen distintos atributos como criterios de consulta. Para responder a estos requisitos, puede crear uno o más índices secundarios globales y emitir solicitudes `Query` en ellos en Amazon DynamoDB.

Temas

- Situación: Uso de un índice secundario global
- Proyecciones de atributos

- Esquema de claves de varios atributos
- Lectura de datos de un índice secundario global
- Sincronización de datos entre tablas e índices secundarios globales
- Clases de tabla con un índice secundario global
- Consideraciones sobre el rendimiento aprovisionado para los índices secundarios globales
- Consideraciones sobre el almacenamiento para los índices secundarios globales
- Patrones de diseño
- Administración de índices secundarios globales en DynamoDB
- Detección y corrección de infracciones de la clave del índice en DynamoDB
- Trabajo con índices secundarios globales: Java
- Trabajar con índices secundarios globales: .NET
- Uso de índices secundarios globales en DynamoDB con la AWS CLI

Situación: Uso de un índice secundario global

Por poner un ejemplo, tomemos una tabla denominada `GameScores` que realiza el seguimiento de los usuarios y las puntuaciones de una aplicación de juegos para móviles. Cada elemento en `GameScores` se identifica por una clave de partición (`UserId`) y una clave de ordenación (`GameTitle`). En el siguiente diagrama se muestra cómo se organizarían los elementos de la tabla. No se muestran todos los atributos.

GameScores

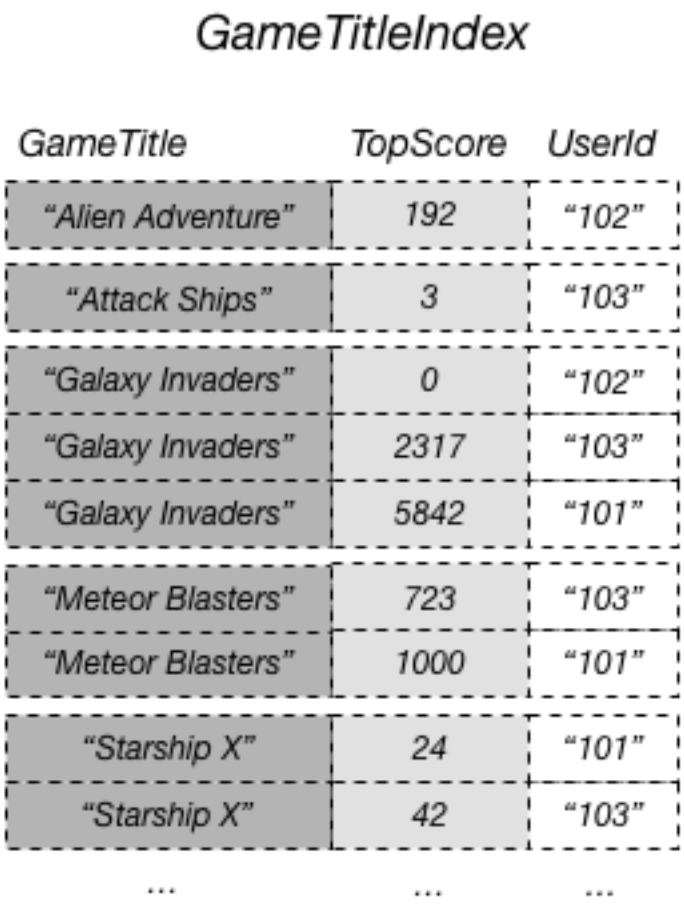
UserId	GameTitle	TopScore	TopScoreDateTime	Wins	Losses	
"101"	"Galaxy Invaders"	5842	"2015-09-15:17:24:31"	21	72	...
"101"	"Meteor Blasters"	1000	"2015-10-22:23:18:01"	12	3	...
"101"	"Starship X"	24	"2015-08-31:13:14:21"	4	9	...
"102"	"Alien Adventure"	192	"2015-07-12:11:07:56"	32	192	...
"102"	"Galaxy Invaders"	0	"2015-09-18:07:33:42"	0	5	...
"103"	"Attack Ships"	3	"2015-10-19:01:13:24"	1	8	...
"103"	"Galaxy Invaders"	2317	"2015-09-11:06:53:00"	40	3	...
"103"	"Meteor Blasters"	723	"2015-10-19:01:13:24"	22	12	...
"103"	"Starship X"	42	"2015-07-11:06:53:00"	4	19	...
...	...	...	...	...	...	

Ahora, supongamos que desea escribir una aplicación de clasificación para mostrar las puntuaciones máximas de cada juego. Una consulta que especifique los atributos de clave (`UserId` y `GameTitle`) sería muy eficiente. Sin embargo, si la aplicación tuviese que recuperar datos de `GameScores` solo según `GameTitle`, tendría que usar una operación `Scan`. A medida que se agregan elementos a la tabla, los exámenes de todos los datos resultarían lentos e ineficientes. En consecuencia, resultaría difícil responder a preguntas como estas:

- ¿Cuál es la puntuación máxima que se ha registrado en el juego Meteor Blasters?
- ¿Qué usuario ha obtenido la mejor puntuación en Galaxy Invaders?
- ¿Cuál es la mayor proporción de partidas ganadas respecto a las perdidas?

Para agilizar las consultas de atributos sin clave, puede crear un índice secundario global. Un índice secundario global contiene una selección de atributos de la tabla base, pero están organizados por una clave principal distinta de la clave principal de la tabla. La clave de índice no requiere disponer de ninguno de los atributos de clave de la tabla. Ni siquiera necesita el mismo esquema de claves que una tabla.

Por ejemplo, puede crear un índice secundario global llamado `GameTitleIndex`, con una clave de partición de `GameTitle` y una clave de ordenación de `TopScore`. Puesto que los atributos de clave principal de la tabla base siempre se proyectan en un índice, el atributo `UserId` también está presente. En el siguiente diagrama se muestra el aspecto que tendría el índice `GameTitleIndex`.



Ahora, puede consultar `GameTitleIndex` y obtener fácilmente las puntuaciones de Meteor Blasters. Los resultados se ordenan según los valores de la clave de ordenación, `TopScore`. Si establece el parámetro `ScanIndexForward` en false, los resultados se devuelven en orden descendente, de modo que la puntuación máxima se devuelve en primer lugar.

Cada índice secundario global debe tener una clave de partición y puede tener una clave de ordenación opcional. El esquema de claves de índice puede ser distinto del de la tabla base. La tabla podría tener una clave principal simple (clave de partición) y, en cambio, se podría crear un índice secundario global con una clave principal compuesta (clave de partición y clave de ordenación), o viceversa. Los atributos de clave del índice pueden constar de cualquier atributo de nivel superior de tipo `String`, `Number` o `Binary` de la tabla base. No se admite ningún otro tipo de escalar, documento ni conjunto.

Puede proyectar otros atributos de la tabla base en el índice si lo desea. Cuando se consulta el índice, DynamoDB puede recuperar estos atributos proyectados de forma eficiente. Sin embargo, las consultas de índice secundario global no permiten recuperar atributos de la tabla base. Por ejemplo, si consulta `GameTitleIndex` como se muestra en el diagrama anterior, la consulta no podría obtener acceso a ningún atributo sin clave excepto a `TopScore` (aunque los atributos de clave `GameTitle` y `UserId` se proyectarían automáticamente).

En una tabla de DynamoDB, cada valor de clave debe ser único. Sin embargo, no es obligatorio que los valores de clave de un índice secundario global sean únicos. A modo de ejemplo, supongamos que un juego denominado Comet Quest es especialmente difícil, de tal forma que muchos usuarios nuevos intentan lograr una puntuación superior a cero, sin conseguirlo. A continuación se muestran algunos datos que podrían representar esta situación.

UserId	GameTitle	TopScore
123	Comet Quest	0
201	Comet Quest	0
301	Comet Quest	0

Cuando estos datos se agregan a la tabla `GameScores`, DynamoDB los propaga a `GameTitleIndex`. Si, a continuación, consultamos el índice con el valor Comet Quest en `GameTitle` y el valor 0 en `TopScore`, se devuelven los valores siguientes.

GameTitle	*TopScore*	*UserId*
"Comet Quest"	*0*	*"123"*
"Comet Quest"	*0*	*"201"*
"Comet Quest"	*0*	*"301"*

Solo aparecerán en la respuesta los elementos que tengan los valores de clave especificados. Dentro de ese conjunto de datos, los elementos no aparecen en ningún orden concreto.

Un índice secundario global solo realiza el seguimiento de los elementos de datos cuyos atributos de clave existen realmente. Por ejemplo, supongamos que ha agregado otro elemento nuevo a la tabla `GameScores`, pero que solo ha proporcionado los atributos de clave principal obligatorios.

UserId	GameTitle
400	Comet Quest

Al no haber especificado el atributo `TopScore`, DynamoDB no propagará este elemento a `GameTitleIndex`. Por lo tanto, si consultase `GameScores` para obtener todos los elementos de Comet Quest, obtendría los siguientes cuatro elementos.

UserId	GameTitle	TopScore
"123"	"Comet Quest"	0
"201"	"Comet Quest"	0
"301"	"Comet Quest"	0
"400"	"Comet Quest"	

Una consulta parecida de `GameTitleIndex` devolvería tres elementos, en lugar de cuatro. Esto se debe a que el elemento cuyo valor de `TopScore` no existe no se propaga al índice.

GameTitle	*TopScore*	*UserId*
"Comet Quest"	*0*	*"123"*
"Comet Quest"	*0*	*"201"*
"Comet Quest"	*0*	*"301"*

Proyecciones de atributos

Una proyección es el conjunto de atributos que se copia de una tabla en un índice secundario. La clave de partición y la clave de ordenación de la tabla siempre se proyectan en el índice; puede proyectar otros atributos para admitir los requisitos de consulta de la aplicación. Cuando consulta un índice, Amazon DynamoDB puede acceder a cualquier atributo de la proyección como si esos atributos estuvieran en una tabla propia.

Al crear un índice secundario, debe especificar los atributos que se proyectarán en el índice. DynamoDB ofrece tres opciones diferentes para esto:

- KEYS_ONLY: cada elemento del índice consta únicamente de los valores de la clave de partición y la clave de ordenación de la tabla, así como de los valores de las claves del índice. La opción `KEYS_ONLY` da como resultado el índice secundario más pequeño posible.
- INCLUDE: además de los atributos que se describen en `KEYS_ONLY`, el índice secundario incluirá otros atributos sin clave que se especifiquen.
- ALL: el índice secundario incluye todos los atributos de la tabla de origen. Debido a que todos los datos de la tabla están duplicados en el índice, un resultado de proyección `ALL` en el índice secundario más grande posible.

En el diagrama anterior, `GameTitleIndex` tiene solo un atributo proyectado: `UserId`. Por tanto, mientras una aplicación puede determinar de forma eficiente el `UserId` de los jugadores que tienen mayor puntuación en cada juego utilizando `GameTitle` y `TopScore` en las consultas, no puede determinar de manera eficiente la mayor proporción de partidas ganadas respecto a las perdidas para los jugadores con mayor puntuación. Para ello, tendría que realizar una consulta adicional en la tabla base para recuperar las partidas ganadas y perdidas de cada uno de los jugadores con mayor puntuación. Una manera más eficiente de consultar estos datos sería proyectar estos atributos de la tabla base en el índice secundario global, tal y como se muestra en este diagrama.

GameTitleIndex

GameTitle	TopScore	UserId	Wins	Losses
"Alien Adventure"	192	"102"	32	192
"Attack Ships"	3	"103"	1	8
"Galaxy Invaders"	0	"102"	0	5
"Galaxy Invaders"	2317	"103"	40	3
"Galaxy Invaders"	5842	"101"	21	72
"Meteor Blasters"	723	"103"	22	12
"Meteor Blasters"	1000	"101"	12	3
"Starship X"	24	"101"	4	9
"Starship X"	42	"103"	4	19
...	...	...	...	...

Dado que los atributos sin clave `Wins` y `Losses` se proyectan en el índice, una aplicación puede determinar la proporción de partidas ganadas respecto a las perdidas en cualquier juego, o para cualquier combinación de juego e identificador de usuario.

Cuando elija los atributos para proyectarlos en un índice secundario global, debe estudiar el equilibrio entre los costes de rendimiento aprovisionado y de almacenamiento:

- Si solo necesita obtener acceso a algunos atributos con la menor latencia posible, puede ser conveniente proyectar solamente esos atributos en un índice secundario global. Cuando menor es el índice, menos cuesta almacenarlo y menos son los costes de escritura.
- Si la aplicación va a obtener acceso con frecuencia a determinados atributos sin clave, puede ser interesante proyectarlos en un índice secundario global. Los costes del almacenamiento adicionales del índice secundario global compensarán el coste que supondrían los frecuentes exámenes de la tabla.

- Si tiene que obtener acceso a la mayoría de los atributos sin clave con frecuencia, puede proyectar estos atributos o, incluso, toda la tabla base, en un índice secundario global. Esto le da la máxima flexibilidad. Sin embargo, el coste del almacenamiento aumentaría y podría llegar a duplicarse.
- Si la aplicación tiene que consultar una tabla con poca frecuencia, pero tiene que realizar gran cantidad de escrituras o actualizaciones en los datos de la tabla, puede ser conveniente proyectar `KEYS_ONLY`. El índice secundario global tendrá el tamaño mínimo, pero estaría disponible siempre que se requiriese para actividades de consulta.

Esquema de claves de varios atributos

Los índices secundarios globales admiten claves de varios atributos, lo que le permite componer claves de partición y claves de clasificación a partir de varios atributos. Con las claves de varios atributos, puede crear una clave de partición a partir de un máximo de cuatro atributos y una clave de clasificación a partir de un máximo de cuatro atributos, hasta un total de ocho atributos por esquema de clave.

Las claves de varios atributos simplifican el modelo de datos al eliminar la necesidad de concatenar manualmente los atributos en claves sintéticas. En lugar de crear cadenas compuestas como `TOURNAMENT#WINTER2024#REGION#NA-EAST`, puede utilizar directamente los atributos naturales del modelo de dominio. DynamoDB gestiona automáticamente la lógica de clave compuesta, agrupando varios atributos de clave de partición para distribuir los datos y manteniendo el orden jerárquico de varios atributos de clave de clasificación.

Por ejemplo, considere un sistema de torneos de juegos en el que desee organizar las partidas por torneo y región. Con las claves de varios atributos, puede definir la clave de partición como dos atributos independientes: `tournamentId` y `region`. Del mismo modo, puede definir la clave de clasificación mediante varios atributos como `round`, `bracket` y `matchId` para crear una jerarquía natural. Este enfoque mantiene los datos mecanografiados y el código limpio, sin manipular ni analizar las cadenas.

Al consultar un índice secundario global con claves de varios atributos, debe especificar todos los atributos de las claves de partición mediante condiciones de igualdad. Para los atributos de las claves de clasificación, puede consultarlos de izquierda a derecha en el orden en que están definidos en el esquema de claves. Esto significa que puede consultar solo el primer atributo de clave de clasificación, los dos primeros atributos juntos o todos los atributos juntos, pero no puede omitir los atributos del medio. Condiciones de desigualdad como >, <, `BETWEEN` o `begins_with()` deben ser la última condición de la consulta.

Las claves de varios atributos funcionan especialmente bien al crear índices secundarios globales en tablas existentes. Puede usar atributos que ya existen en la tabla sin rellenar los datos con claves sintéticas. Esto facilita la agregación de nuevos patrones de consulta a la aplicación mediante la creación de índices que reorganizan los datos mediante diferentes combinaciones de atributos.

Cada atributo de una clave de varios atributos puede tener su propio tipo de datos: `String` (S), `Number` (N) o `Binary` (B). Al elegir los tipos de datos, tenga en cuenta que los atributos de `Number` se ordenan numéricamente sin necesidad de rellenar ceros, mientras que los atributos de `String` se ordenan lexicográficamente. Por ejemplo, si utiliza un tipo de `Number` para un atributo de puntuación, los valores 5, 50, 500 y 1000 se ordenan en orden numérico natural. Los mismos valores que el tipo de `String` se ordenarían como “1000”, “5”, “50” o “500”, a menos que los rellene con ceros a la izquierda.

Al diseñar claves de varios atributos, ordene los atributos del más general al más específico. En el caso de las claves de partición, combine los atributos que siempre se consulten juntos y que proporcionen una buena distribución de los datos. En el caso de las claves de clasificación, coloque los atributos consultados con frecuencia en primer lugar de la jerarquía para maximizar la flexibilidad de las consultas. Este orden le permite realizar consultas en cualquier nivel de detalle que coincida con los patrones de acceso.

Consulte los ejemplos de Claves de varios atributos para implementación.

Lectura de datos de un índice secundario global

Puede recuperar elementos de un índice secundario global mediante las operaciones `Query` y `Scan`. Las opreciones `GetItem` y `BatchGetItem` no se pueden usar en un índice secundario global.

Consulta a un índice secundario global

Puede utilizar la operación `Query` para obtener acceso a uno o varios elementos de un índice secundario global. En la consulta se debe especificar el nombre de la tabla base y el nombre del índice que se desea utilizar, los atributos que se devolverán en los resultados de la consulta y las condiciones de consulta que se van a aplicar. DynamoDB puede devolver los resultados en orden ascendente o descendente.

Tomemos los datos siguientes devueltos por una operación `Query` que solicita datos de juegos para una aplicación de clasificación de juegos.

```
{
    "TableName": "GameScores",
```

```
    "IndexName": "GameTitleIndex",
    "KeyConditionExpression": "GameTitle = :v_title",
    "ExpressionAttributeValues": {
        ":v_title": {"S": "Meteor Blasters"}
    },
    "ProjectionExpression": "UserId, TopScore",
    "ScanIndexForward": false
}
```

En esta consulta:

- DynamoDB accede GameTitleIndex utilizando la clave de partición GameTitle para localizar los elementos de índice correspondientes a Meteor Blasters. Todos los elementos de índice que tienen esta clave se almacenan en posiciones adyacentes, para agilizar su recuperación.
- En este juego, DynamoDB utiliza el índice para obtener acceso a todos los identificadores de los usuarios y a las puntuaciones máximas de este juego.
- Los resultados se devuelven ordenados por orden descendente, porque el parámetro `ScanIndexForward` se ha establecido en false.

Análisis de un índice secundario global

Puede usar la operación `Scan` para recuperar todos los datos de un índice secundario global. Debe proporcionar el nombre de la tabla base y el nombre del índice en la solicitud. Con una operación `Scan`, DynamoDB lee todos los datos del índice y los devuelve a la aplicación. También puede solicitar que solo se devuelvan algunos de los datos y se descarten los demás. Para ello, se utiliza el parámetro `FilterExpression` de la operación `Scan`. Para obtener más información, consulte Expresiones de filtro para el análisis.

Sincronización de datos entre tablas e índices secundarios globales

DynamoDB sincroniza automáticamente cada índice secundario global con su tabla base. Cuando una aplicación escribe o elimina elementos en una tabla, cualquier índice secundario global de esa tabla se actualizan de forma asincrónica, aplicando un modelo de consistencia final. Las aplicaciones nunca escriben directamente en un índice. Sin embargo, es importante que comprenda las implicaciones de cómo DynamoDB mantiene estos índices.

Los índices secundarios globales heredan el modo de capacidad de lectura/escritura de la tabla base. Para obtener más información, consulte Aspectos a tener en cuenta al cambiar los modos de capacidad en DynamoDB.

Al crear un índice secundario global, debe especificar uno o varios atributos de clave de índice y sus tipos de datos. Esto significa que, cada vez que se escribe un elemento en la tabla base, los tipos de datos de esos atributos deben coincidir con los tipos de datos del esquema de claves de índice. En el caso de `GameTitleIndex`, el tipo de datos de la clave de partición `GameTitle` del índice es `String`. La clave de ordenación `TopScore` del índice es de tipo `Number`. Si intenta agregar un elemento a la tabla `GameScores`, pero especifica un tipo de datos distinto para `GameTitle` o `TopScore`, DynamoDB devolverá una excepción `ValidationException`, porque los tipos de datos no coinciden.

Al colocar o eliminar elementos en una tabla, sus índices secundarios globales se actualizan de forma consistente final. En condiciones normales, los cambios en los datos de la tabla se propagan a los índices secundarios globales casi al instante. No obstante, en algunos escenarios de error improbables, pueden producirse retardos de propagación más prolongados. Debido a ello, las aplicaciones deben prever y controlar las situaciones en las que una consulta de un índice secundario global devuelva resultados que no se encuentren actualizados.

Si escribe un elemento en una tabla, no tiene que especificar los atributos de ninguna clave de ordenación del índice secundario global. Si utilizamos `GameTitleIndex` a modo de ejemplo, no sería preciso especificar un valor para el atributo `TopScore` para poder escribir un nuevo elemento en la tabla `GameScores`. En este caso, DynamoDB no escribe ningún dato en el índice para este elemento concreto.

Una tabla con muchos índices secundarios globales devengará costes más elevados por la actividad de escritura que las tablas con menos índices. Para obtener más información, consulte Consideraciones sobre el rendimiento aprovisionado para los índices secundarios globales.

Clases de tabla con un índice secundario global

Un índice secundario global siempre utilizará la misma clase de tabla que su tabla base. Cada vez que se agrega un nuevo índice secundario global para una tabla, el nuevo índice utilizará la misma clase de tabla que su tabla base. Cuando se actualiza la clase de tabla de una tabla, también se actualizan todos los índices secundarios globales asociados.

Consideraciones sobre el rendimiento aprovisionado para los índices secundarios globales

Al crear un índice secundario global en una tabla en modo aprovisionado, debe especificar las unidades de capacidad de lectura y escritura para la carga de trabajo prevista de ese índice. Los

ajustes de rendimiento aprovisionado de un índice secundario global son independientes de los de su tabla base. Una operación `Query` en un consume unidades de capacidad de lectura del índice, no de la tabla base. Al colocar, actualizar o eliminar elementos en una tabla, sus índices secundarios globales se actualizan de forma consistente final. Estas actualizaciones de índices consumen unidades de capacidad de escritura del índice, no de la tabla base.

Por ejemplo, si realiza una operación `Query` en un índice secundario global y supera su capacidad de lectura aprovisionada, la solicitud se someterá a una limitación controlada. Si realiza una intensa actividad de escritura en la tabla, pero un índice secundario global de esa tabla no tiene suficiente capacidad de escritura, entonces la actividad de escritura de la tabla se someterá a una limitación controlada.

> **Important**
>
> Para evitar una posible limitación controlada, la capacidad de escritura provisionada para un índice secundario global debe ser igual o mayor que la capacidad de escritura de la tabla base, ya que las nuevas actualizaciones se escribirán tanto en la tabla base como en el índice secundario global.

Para modificar los ajustes de rendimiento aprovisionado de un índice secundario global, use la operación `DescribeTable`. Se devuelve información detallada sobre cada índices secundario global de la tabla.

Unidades de capacidad de lectura

Los índices secundarios globales admiten las lecturas consistentes finales, cada una de la cuales consume la mitad de una unidad de capacidad de lectura. Esto quiere decir que una sola consulta en un índice secundario global permite recuperar hasta 2 × 4 = 8 KB por unidad de capacidad de lectura.

Para las consultas en un índce secundario global, DynamoDB calcula la actividad de lectura provisionada de la misma forma que para las consultas en una tabla. La única diferencia es que el cálculo se basa en el tamaño de las entradas del índice, no en el tamaño del elemento en la tabla base. El número de unidades de capacidad de lectura es la suma de todos los tamaños de los atributos proyectados para todos los elementos devueltos. El resultado se redondea al múltiplo de 4 KB inmediatamente superior. Para obtener más información sobre cómo calcula DynamoDB el consumo de rendimiento aprovisionado, consulte Modo de capacidad aprovisionada de DynamoDB.

El tamaño máximo de los resultados que devuelve una operación `Query` es de 1 MB. Esta cifra incluye los tamaños de todos los nombres y valores de los atributos de todos los elementos devueltos.

Por ejemplo, tomemos un índice secundario global en el que cada elemento contiene 2000 bytes de datos. Ahora, supongamos que se utiliza una operación `Query` en este índice y que la consulta `KeyConditionExpression` devuelve ocho elementos. El tamaño total de los elementos coincidentes es de 2000 bytes x 8 elementos = 16 000 bytes. El resultado se redondea al múltiplo de 4 KB inmediatamente superior. Puesto que las consultas en un índice secundario global presentan consistencia final, el coste total es de 0,5 (16 KB / 4 KB), lo que equivale a 2 unidades de capacidad de lectura.

Unidades de capacidad de escritura

Cuando se agrega, actualiza o elimina un elemento de una tabla y esto afecta a un índice secundario global, este índice secundario global consume unidades de capacidad de escritura provisionadas por esta operación. El coste total de desempeño provisionado de una escritura es la suma de las unidades de capacidad de escritura consumidas al escribir en la tabla base y aquellas consumidas al actualizar los índices secundarios globales. Si una escritura en una tabla no requiere que se actualice un índice secundario global, no se consume ninguna capacidad de escritura del índice.

Para que una escritura en una tabla se lleve a cabo correctamente, la capacidad de desempeño provisionada definida para la tabla y todos sus índices secundarios globales debe ser suficiente para admitir esa escritura. De lo contrario, la escritura en la tabla se someterá a una limitación controlada.

> **Important**
>
> Al crear un índice secundario global (GSI), las operaciones de escritura en la tabla base se pueden limitar si la actividad del GSI resultante de las escrituras en la tabla base supera la capacidad de escritura aprovisionada del GSI. Esta limitación afecta a todas las operaciones de escritura, desde el proceso de indexación hasta la posible interrupción de las cargas de trabajo de producción. Para obtener más información, consulte Solución de problemas de limitación en Amazon DynamoDB.

El coste de escribir un elemento en un índice secundario global depende de varios factores:

- Si escribe un nuevo elemento en la tabla que define un atributo indexado o actualiza un elemento existente para definir un atributo indexado no definido previamente, se requiere una operación de escritura para colocar el elemento en el índice.
- Si al actualizar la tabla se cambia el valor de un atributo de clave indexado (de A a B), se requieren dos escrituras, una para eliminar el elemento anterior del índice y otra para colocar el nuevo elemento en el índice.
- Si ya hay un elemento en el índice, pero al escribir en la tabla se elimina el atributo indexado, se requiere una escritura para eliminar la proyección del elemento anterior del índice.
- Si no hay ningún elemento presente en el índice antes o después de actualizar el elemento, no se devenga ningún coste de escritura adicional para el índice.
- Si al actualizar la tabla solo se cambia el valor de los atributos proyectados en el esquema de claves del índice, pero no se cambia el valor de ningún atributo de clave indexado, se requiere una escritura para actualizar los valores de los atributos proyectados en el índice.

Al calcular las unidades de capacidad de escritura, en todos estos factores se presupone que el tamaño de cada elemento del índice es menor o igual que el tamaño de elemento de 1 KB. Las entradas de índice de mayor tamaño requieren más unidades de capacidad de escritura. Para minimizar los costos de escritura, es conveniente estudiar qué atributos tendrán que devolver las consultas y proyectar solamente esos atributos en el índice.

Consideraciones sobre el almacenamiento para los índices secundarios globales

Cuando una aplicación escribe un elemento en una tabla, DynamoDB copia automáticamente el subconjunto de atributos correcto en todos los índices secundarios globales en los que deban aparecer esos atributos. Se aplica un cargo en su cuenta de AWS por el almacenamiento del elemento en la tabla base y también por el almacenamiento de los atributos en todos los índices secundarios globales de esa tabla.

La cantidad de espacio utilizado por un elemento de índice es la suma de lo siguiente:

- El tamaño en bytes de la clave principal (clave de partición y clave de ordenación) de la tabla base
- El tamaño en bytes del atributo de clave de índice
- El tamaño en bytes de los atributos proyectados (si procede)
- 100 bytes de gastos generales por cada elemento de índice

Para calcular los requisitos de almacenamiento de un índice secundario global, puede calcular el tamaño medio de un elemento del índice y, a continuación, multiplicarlo por el número de elementos de la tabla base que tienen atributos de clave del índice secundario global.

Si una tabla contiene un elemento en el que no se han definido atributos determinados, pero ese atributo se ha definido como clave de partición o clave de clasificación del índice, DynamoDB no escribe ningún dato para ese elemento en el índice.

Patrones de diseño

Los patrones de diseño ofrecen soluciones probadas a los desafíos más comunes cuando se trabaja con índices secundarios globales. Estos patrones le ayudan a crear aplicaciones eficientes y escalables, ya que le muestran cómo estructurar los índices para casos de uso específicos.

Cada patrón incluye una guía de implementación completa con ejemplos de código, prácticas recomendadas y casos de uso reales que le ayudarán a aplicar el patrón a sus propias aplicaciones.

Temas

- Patrón de claves de varios atributos

Patrón de claves de varios atributos

Descripción general

Las claves de varios atributos le permiten crear claves de partición y clasificación del índice secundario global (GSI) compuestas por un máximo de cuatro atributos cada una. Esto reduce el código del cliente y facilita el modelado inicial de los datos y la posterior agregación de nuevos patrones de acceso.

Pensemos en un escenario común: para crear un GSI que consulte los elementos por varios atributos jerárquicos, tradicionalmente se necesitaría crear claves sintéticas concatenando valores. Por ejemplo, en una aplicación de juegos, para consultar las partidas de un torneo por torneo, región y ronda, puede crear una clave de partición GSI sintética, como TOURNAMENT#WINTER2024#REGION#NA-EAST y una clave de clasificación sintética, como ROUND#SEMIFINALS#BRACKET#UPPER. Este enfoque funciona, pero requiere la concatenación de cadenas al escribir datos, el análisis al leer y el relleno de claves sintéticas en todos los elementos existentes si va a agregar el GSI a una tabla existente. Esto hace que el código sea más desordenado y difícil de mantener la seguridad tipográfica en los componentes clave individuales.

Las claves de varios atributos resuelven este problema para los GSI. La clave de partición de GSI se define mediante varios atributos existentes, como el tournamentId y la región. DynamoDB gestiona automáticamente la lógica de claves compuestas y las agrupa para la distribución de los datos. Los elementos se escriben con los atributos naturales del modelo de dominio y el GSI los indexa automáticamente. Sin concatenación, sin análisis, sin relleno. El código se mantiene limpio, los datos se mantienen escritos y las consultas se mantienen sencillas. Este enfoque resulta especialmente útil cuando se dispone de datos jerárquicos con agrupaciones de atributos naturales (como torneo → región → ronda u organización → departamento → equipo).

Ejemplo de aplicación

Esta guía explica cómo crear un sistema de seguimiento de partidos de torneos para una plataforma de deportes electrónicos. La plataforma necesita consultar los partidos de manera eficiente en múltiples dimensiones: por torneo y región para administrar los grupos, por jugador para ver el historial de partidos y por fecha para programarlos.

Modelo de datos

En este tutorial, el sistema de seguimiento de los partidos del torneo admite tres patrones de acceso principales, cada uno de los cuales requiere una estructura de clave diferente:

Patrón de acceso 1: busque un partido específico por su ID único

- Solución: tabla base con `matchId` como clave de partición

Patrón de acceso 2: consulte todos los partidos de un torneo y una región específicos y, si lo prefiere, filtre por ronda, cuadro o partido

- Solución: índice secundario global con clave de partición de varios atributos (`tournamentId` + `region`) y clave de clasificación de varios atributos (`round` + `bracket` + `matchId`)
- Consultas de ejemplo: “Todos los partidos de WINTER2024 en la región NA-EAST” o “Todos los partidos de SEMIFINALES en el corchete SUPERIOR para WINTER2024/NA-EAST”

Patrón de acceso 3: consulte el historial de partidos de un jugador o, si lo prefiere, filtre por intervalo de fechas o por ronda del torneo

- Solución: índice secundario global con clave de partición única (`player1Id`) y clave de clasificación de varios atributos (`matchDate` + `round`)

- Consultas de ejemplo: “Todos los partidos del jugador 101” o “Los partidos del jugador 101 en enero de 2024”

La diferencia clave entre el enfoque tradicional y el de varios atributos queda clara al examinar la estructura de los elementos:

Enfoque de índice secundario global tradicional (claves concatenadas):

```
// Manual concatenation required for GSI keys
const item = {
    matchId: 'match-001',                                          // Base table PK
    tournamentId: 'WINTER2024',
    region: 'NA-EAST',
    round: 'SEMIFINALS',
    bracket: 'UPPER',
    player1Id: '101',
    // Synthetic keys needed for GSI
    GSI_PK: `TOURNAMENT#${tournamentId}#REGION#${region}`,       // Must concatenate
    GSI_SK: `${round}#${bracket}#${matchId}`,                    // Must concatenate
    // ... other attributes
};
```

Enfoque de índice secundario global de varios atributos (claves nativas):

```
// Use existing attributes directly - no concatenation needed
const item = {
    matchId: 'match-001',                                          // Base table PK
    tournamentId: 'WINTER2024',
    region: 'NA-EAST',
    round: 'SEMIFINALS',
    bracket: 'UPPER',
    player1Id: '101',
    matchDate: '2024-01-18',
    // No synthetic keys needed - GSI uses existing attributes directly
    // ... other attributes
};
```

Con las claves de varios atributos, los elementos se escriben una sola vez con atributos de dominio naturales. DynamoDB los indexa automáticamente en varios GSI sin necesidad de claves concatenadas sintéticas.

Esquema de tabla base:

- Clave de partición: `matchId` (1 atributo)

Esquema de índice secundario global (TournamentRegionIndex con claves de varios atributos):

- Clave de partición: `tournamentId`, `region` (2 atributos)
- Clave de clasificación: `round`, `bracket`, `matchId` (3 atributos)

Esquema de índice secundario global (PlayerMatchHistoryIndex con claves de varios atributos):

- Clave de partición: `player1Id` (1 atributo)
- Clave de clasificación: `matchDate`, `round` (2 atributos)

Tabla base: TournamentMatches

matchId (PK)	tourname tId	region	round	soporte	player1Id	player2Id	matchDa	ganador	puntuació n
match-0(	WINTER 4	NA-EAST	FINALES	CAMPEO MUNDIA	101	103	2024-01-0	101	3-1
match-0(	WINTER 4	NA-EAST	SEMIFIN ES	UPPER	101	105	2024-01-8	101	3-2
match-0(	WINTER 4	NA-EAST	SEMIFIN ES	UPPER	103	107	2024-01-8	103	3-0
match-0(	WINTER 4	NA-EAST	CUARTO DE FINAL	UPPER	101	109	2024-01-5	101	3-1
match-0(	WINTER 4	NA-WEST	FINALES	CAMPEO MUNDIA	102	104	2024-01-0	102	3-2

matchId (PK)	tourname tId	region	round	soporte	player1Id	player2Id	matchDa	ganador	puntuació n
match-0(	WINTER 4	NA-WEST	SEMIFIN ES	UPPER	102	106	2024-01-8	102	3-1
match-0(	SPRING 4	NA-EAST	CUARTC DE FINAL	UPPER	101	108	2024-03-5	101	3-0
match-0(	SPRING 4	NA-EAST	CUARTC DE FINAL	LOWER	103	110	2024-03-5	103	3-2

GSI: TournamentRegionIndex (claves de varios atributos)

tourname tId (PK)	región (PK)	ronda (SK)	soporte (SK)	matchId (SK)	player1Id	player2Id	matchDa	ganador	puntuació n
WINTER 4	NA-EAST	FINALES	CAMPEC O MUNDIA	match-0(	101	103	2024-01-0	101	3-1
WINTER 4	NA-EAST	CUARTC DE FINAL	UPPER	match-0(	101	109	2024-01-5	101	3-1
WINTER 4	NA-EAST	SEMIFIN ES	UPPER	match-0(	101	105	2024-01-8	101	3-2
WINTER 4	NA-EAST	SEMIFIN ES	UPPER	match-0(	103	107	2024-01-8	103	3-0
WINTER 4	NA-WEST	FINALES	CAMPEC O MUNDIA	match-0(	102	104	2024-01-0	102	3-2

tourname tId (PK)	región (PK)	ronda (SK)	soporte (SK)	matchId (SK)	player1Id	player2Id	matchDa	ganador	puntuació n
WINTER 4	NA-WEST	SEMIFIN ES	UPPER	match-0(	102	106	2024-01-8	102	3-1
SPRING 4	NA-EAST	CUARTC DE FINAL	LOWER	match-0(	103	110	2024-03-5	103	3-2
SPRING 4	NA-EAST	CUARTC DE FINAL	UPPER	match-0(	101	108	2024-03-5	101	3-0

GSI: PlayerMatchHistoryIndex (claves de varios atributos)

player1Id (PK)	matchDa (SK)	ronda (SK)	tourname tId	region	soporte	matchId	player2Id	ganador	puntuació n
101	2024-01-5	CUARTC DE FINAL	WINTER 4	NA-EAST	UPPER	match-0(	109	101	3-1
101	2024-01-8	SEMIFIN ES	WINTER 4	NA-EAST	UPPER	match-0(	105	101	3-2
101	2024-01-0	FINALES	WINTER 4	NA-EAST	CAMPEC O MUNDIA	match-0(	103	101	3-1
101	2024-03-5	CUARTC DE FINAL	SPRING 4	NA-EAST	UPPER	match-0(	108	101	3-0
102	2024-01-8	SEMIFIN ES	WINTER 4	NA-WEST	UPPER	match-0(	106	102	3-1

player1Id (PK)	matchDa (SK)	ronda (SK)	tourname tId	region	soporte	matchId	player2Id	ganador	puntuació n
102	2024-01-0	FINALES	WINTER 4	NA-WEST	CAMPEO MUNDIA	match-0(	104	102	3-2
103	2024-01-8	SEMIFIN ES	WINTER 4	NA-EAST	UPPER	match-0(	107	103	3-0
103	2024-03-5	CUARTO DE FINAL	SPRING 4	NA-EAST	LOWER	match-0(	110	103	3-2

Requisitos previos

Antes de comenzar, asegúrese de que dispone de lo siguiente:

Permisos y cuenta

- Una cuenta de AWS activa (cree una aquí si es necesario)
- Permisos de IAM para las operaciones de DynamoDB:
  - `dynamodb:CreateTable`
  - `dynamodb:DeleteTable`
  - `dynamodb:DescribeTable`
  - `dynamodb:PutItem`
  - `dynamodb:Query`
  - `dynamodb:BatchWriteItem`

> **Note**
>
> Nota de seguridad: para uso de producción, cree una política de IAM personalizada con solo los permisos que necesite. Para este tutorial, puede usar la política administrada de AWS `AmazonDynamoDBFullAccessV2`.

Entorno de desarrollo

- Node.js se ha instalado en el equipo
- Credenciales de AWS configuradas mediante uno de estos métodos:

Opción 1: CLI de AWS

```
aws configure
```

Opción 2: Variables de entorno

```
export AWS_ACCESS_KEY_ID=your_access_key_here
export AWS_SECRET_ACCESS_KEY=your_secret_key_here
export AWS_DEFAULT_REGION=us-east-1
```

Instalación de los paquetes obligatorios

```
npm install @aws-sdk/client-dynamodb @aws-sdk/lib-dynamodb
```

Implementación

Paso 1: creación de una tabla con los GSI mediante claves de varios atributos

Cree una tabla con una estructura de clave básica simple y con GSI que utilicen claves de varios atributos.

Ejemplo de código

```
import { DynamoDBClient, CreateTableCommand } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });

const response = await client.send(new CreateTableCommand({
    TableName: 'TournamentMatches',

    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'matchId', KeyType: 'HASH' }             // Simple PK
    ],

    AttributeDefinitions: [
```

```
        { AttributeName: 'matchId', AttributeType: 'S' },
        { AttributeName: 'tournamentId', AttributeType: 'S' },
        { AttributeName: 'region', AttributeType: 'S' },
        { AttributeName: 'round', AttributeType: 'S' },
        { AttributeName: 'bracket', AttributeType: 'S' },
        { AttributeName: 'player1Id', AttributeType: 'S' },
        { AttributeName: 'matchDate', AttributeType: 'S' }
    ],

    // GSIs with multi-attribute keys
    GlobalSecondaryIndexes: [
        {
            IndexName: 'TournamentRegionIndex',
            KeySchema: [
                { AttributeName: 'tournamentId', KeyType: 'HASH' },    // GSI PK
 attribute 1
                { AttributeName: 'region', KeyType: 'HASH' },          // GSI PK
 attribute 2
                { AttributeName: 'round', KeyType: 'RANGE' },          // GSI SK
 attribute 1
                { AttributeName: 'bracket', KeyType: 'RANGE' },        // GSI SK
 attribute 2
                { AttributeName: 'matchId', KeyType: 'RANGE' }         // GSI SK
 attribute 3
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'PlayerMatchHistoryIndex',
            KeySchema: [
                { AttributeName: 'player1Id', KeyType: 'HASH' },       // GSI PK
                { AttributeName: 'matchDate', KeyType: 'RANGE' },      // GSI SK
 attribute 1
                { AttributeName: 'round', KeyType: 'RANGE' }           // GSI SK
 attribute 2
            ],
            Projection: { ProjectionType: 'ALL' }
        }
    ],

    BillingMode: 'PAY_PER_REQUEST'
}));
```

```
console.log("Table with multi-attribute GSI keys created successfully");
```

Decisiones de diseño de clave:

Tabla base: la tabla base utiliza una clave de partición de `matchId` simple para las búsquedas de coincidencias directas, lo que mantiene la estructura de la tabla base sencilla, mientras que los GSI proporcionan patrones de consulta complejos.

Índice secundario global TournamentRegionIndex: el índice secundario global `TournamentRegionIndex` utiliza `tournamentId` + `region` como clave de partición con varios atributos, lo que aísla la región del torneo, ya que los datos se distribuyen mediante el hash de ambos atributos combinados, lo que permite realizar consultas eficaces dentro de un contexto específico de una región o torneo. La clave de clasificación de varios atributos (`round` + `bracket` + `matchId`) proporciona una clasificación jerárquica que permite realizar consultas en cualquier nivel de la jerarquía con un orden natural, desde general (ronda) hasta específico (ID de partido).

Índice secundario global PlayerMatchHistoryIndex: el índice secundario global `PlayerMatchHistoryIndex` reorganiza los datos por jugador usando `player1Id` como clave de partición, lo que permite realizar consultas entre torneos para un jugador específico. La clave de clasificación con varios atributos (`matchDate` + `round`) proporciona un orden cronológico y permite filtrar por intervalos de fechas o rondas específicas del torneo.

Paso 2: insertar datos con atributos nativos

Agrega los datos de los partidos del torneo con atributos naturales. El GSI indexará automáticamente estos atributos sin necesidad de claves sintéticas.

Ejemplo de código

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, PutCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Tournament match data - no synthetic keys needed for GSIs
const matches = [
    // Winter 2024 Tournament, NA-EAST region
    {
        matchId: 'match-001',
        tournamentId: 'WINTER2024',
        region: 'NA-EAST',
```

```
        round: 'FINALS',
        bracket: 'CHAMPIONSHIP',
        player1Id: '101',
        player2Id: '103',
        matchDate: '2024-01-20',
        winner: '101',
        score: '3-1'
    },
    {
        matchId: 'match-002',
        tournamentId: 'WINTER2024',
        region: 'NA-EAST',
        round: 'SEMIFINALS',
        bracket: 'UPPER',
        player1Id: '101',
        player2Id: '105',
        matchDate: '2024-01-18',
        winner: '101',
        score: '3-2'
    },
    {
        matchId: 'match-003',
        tournamentId: 'WINTER2024',
        region: 'NA-EAST',
        round: 'SEMIFINALS',
        bracket: 'UPPER',
        player1Id: '103',
        player2Id: '107',
        matchDate: '2024-01-18',
        winner: '103',
        score: '3-0'
    },
    {
        matchId: 'match-004',
        tournamentId: 'WINTER2024',
        region: 'NA-EAST',
        round: 'QUARTERFINALS',
        bracket: 'UPPER',
        player1Id: '101',
        player2Id: '109',
        matchDate: '2024-01-15',
        winner: '101',
        score: '3-1'
    },
```

```
    // Winter 2024 Tournament, NA-WEST region
    {
        matchId: 'match-005',
        tournamentId: 'WINTER2024',
        region: 'NA-WEST',
        round: 'FINALS',
        bracket: 'CHAMPIONSHIP',
        player1Id: '102',
        player2Id: '104',
        matchDate: '2024-01-20',
        winner: '102',
        score: '3-2'
    },
    {
        matchId: 'match-006',
        tournamentId: 'WINTER2024',
        region: 'NA-WEST',
        round: 'SEMIFINALS',
        bracket: 'UPPER',
        player1Id: '102',
        player2Id: '106',
        matchDate: '2024-01-18',
        winner: '102',
        score: '3-1'
    },

    // Spring 2024 Tournament, NA-EAST region
    {
        matchId: 'match-007',
        tournamentId: 'SPRING2024',
        region: 'NA-EAST',
        round: 'QUARTERFINALS',
        bracket: 'UPPER',
        player1Id: '101',
        player2Id: '108',
        matchDate: '2024-03-15',
        winner: '101',
        score: '3-0'
    },
    {
        matchId: 'match-008',
        tournamentId: 'SPRING2024',
        region: 'NA-EAST',
```

```
        round: 'QUARTERFINALS',
        bracket: 'LOWER',
        player1Id: '103',
        player2Id: '110',
        matchDate: '2024-03-15',
        winner: '103',
        score: '3-2'
    }
];

// Insert all matches
for (const match of matches) {
    await docClient.send(new PutCommand({
        TableName: 'TournamentMatches',
        Item: match
    }));

    console.log(`Added: ${match.matchId} - ${match.tournamentId}/${match.region} -
 ${match.round} ${match.bracket}`);
}

console.log(`\nInserted ${matches.length} tournament matches`);
console.log("No synthetic keys created - GSIs use native attributes automatically");
```

Explicación de la estructura de datos:

Uso de atributos naturales: cada atributo representa un concepto de torneo real sin necesidad de concatenar cadenas ni analizar, lo que proporciona una asignación directa al modelo de dominio.

Indexación automática del índice secundario global: los GSI indexan automáticamente los elementos utilizando los atributos existentes (`tournamentId`, `region`, `round`, `bracket`, `matchId` para TournamentRegionIndex y `player1Id`, `matchDate`, `round` para PlayerMatchHistoryIndex) sin necesidad de claves concatenadas sintéticas.

No es necesario rellenar: al agregar un nuevo índice secundario global con claves de varios atributos a una tabla existente, DynamoDB indexa automáticamente todos los elementos existentes con sus atributos naturales, sin necesidad de actualizar los elementos con claves sintéticas.

Paso 3: consulte el índice secundario global de TournamentRegionIndex con todos los atributos de las claves de partición

En este ejemplo, se consulta el índice secundario global TournamentRegionIndex, que tiene una clave de partición de varios atributos (`tournamentId` + `region`). Todos los atributos de la clave

de partición se deben especificar con condiciones de igualdad en las consultas; no se pueden consultar solo con `tournamentId` ni utilizar operadores de desigualdad en los atributos de la clave de partición.

Ejemplo de código

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, QueryCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Query GSI: All matches for WINTER2024 tournament in NA-EAST region
const response = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region',
    ExpressionAttributeNames: {
        '#region': 'region',  // 'region' is a reserved keyword
        '#tournament': 'tournament'
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST'
    }
}));

console.log(`Found ${response.Items.length} matches for WINTER2024/NA-EAST:\n`);
response.Items.forEach(match => {
    console.log(`  ${match.round} | ${match.bracket} | ${match.matchId}`);
    console.log(`    Players: ${match.player1Id} vs ${match.player2Id}`);
    console.log(`    Winner: ${match.winner}, Score: ${match.score}\n`);
});
```

Resultado previsto:

```
Found 4 matches for WINTER2024/NA-EAST:

  FINALS | CHAMPIONSHIP | match-001
    Players: 101 vs 103
    Winner: 101, Score: 3-1

  QUARTERFINALS | UPPER | match-004
```

```
    Players: 101 vs 109
    Winner: 101, Score: 3-1

  SEMIFINALS | UPPER | match-002
    Players: 101 vs 105
    Winner: 101, Score: 3-2

  SEMIFINALS | UPPER | match-003
    Players: 103 vs 107
    Winner: 103, Score: 3-0
```

Consultas no válidas:

```
// Missing region attribute
KeyConditionExpression: 'tournamentId = :tournament'

// Using inequality on partition key attribute
KeyConditionExpression: 'tournamentId = :tournament AND #region > :region'
```

Rendimiento: las claves de partición de varios atributos se codifican entre sí, lo que proporciona el mismo rendimiento de búsqueda O(1) que las claves de un solo atributo.

Paso 4: consulte las claves de clasificación del índice secundario global de izquierda a derecha

Los atributos de clave de clasificación se deben consultar de izquierda a derecha en el orden en que están definidos en el índice secundario global. En este ejemplo, se muestra cómo consultar el TournamentRegionIndex en diferentes niveles jerárquicos: filtrando solo por `round`, por `round` + `bracket`, o por los tres atributos de clave de clasificación. No puede omitir los atributos que están en el centro; por ejemplo, no puede consultar `round` y `matchId` mientras se omite `bracket`.

Ejemplo de código

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, QueryCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Query 1: Filter by first sort key attribute (round)
console.log("Query 1: All SEMIFINALS matches");
const query1 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
```

```
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 = :round',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'SEMIFINALS'
    }
}));
console.log(`  Found ${query1.Items.length} matches\n`);

// Query 2: Filter by first two sort key attributes (round + bracket)
console.log("Query 2: SEMIFINALS UPPER bracket matches");
const query2 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 = :round AND bracket = :bracket',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'SEMIFINALS',
        ':bracket': 'UPPER'
    }
}));
console.log(`  Found ${query2.Items.length} matches\n`);

// Query 3: Filter by all three sort key attributes (round + bracket + matchId)
console.log("Query 3: Specific match in SEMIFINALS UPPER bracket");
const query3 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 = :round AND bracket = :bracket AND matchId = :matchId',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
```

```
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'SEMIFINALS',
        ':bracket': 'UPPER',
        ':matchId': 'match-002'
    }
}));
console.log(`  Found ${query3.Items.length} matches\n`);

// Query 4: INVALID - skipping round
console.log("Query 4: Attempting to skip first sort key attribute (WILL FAIL)");
try {
    const query4 = await docClient.send(new QueryCommand({
        TableName: 'TournamentMatches',
        IndexName: 'TournamentRegionIndex',
        KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND
 bracket = :bracket',
        ExpressionAttributeNames: {
            '#region': 'region'  // 'region' is a reserved keyword
        },
        ExpressionAttributeValues: {
            ':tournament': 'WINTER2024',
            ':region': 'NA-EAST',
            ':bracket': 'UPPER'
        }
    }));
} catch (error) {
    console.log(`  Error: ${error.message}`);
    console.log(`  Cannot skip sort key attributes - must query left-to-right\n`);
}
```

Resultado previsto:

```
Query 1: All SEMIFINALS matches
  Found 2 matches

Query 2: SEMIFINALS UPPER bracket matches
  Found 2 matches

Query 3: Specific match in SEMIFINALS UPPER bracket
  Found 1 matches

Query 4: Attempting to skip first sort key attribute (WILL FAIL)
```

```
  Error: Query key condition not supported
  Cannot skip sort key attributes - must query left-to-right
```

Reglas de consulta de izquierda a derecha: debe consultar los atributos en orden de izquierda a derecha, sin omitir ninguno.

Patrones validos:

- Solo el primer atributo: `round = 'SEMIFINALS'`
- Los dos primeros atributos: `round = 'SEMIFINALS' AND bracket = 'UPPER'`
- Los tres atributos: `round = 'SEMIFINALS' AND bracket = 'UPPER' AND matchId = 'match-002'`

Patrones no válidos:

- Omisión del primer atributo: `bracket = 'UPPER'` (se salta)
- La consulta está fuera de orden: `matchId = 'match-002' AND round = 'SEMIFINALS'`
- Dejar huecos: `round = 'SEMIFINALS' AND matchId = 'match-002'` (se salta el corchete)

> **Note**
>
> Consejo de diseño: ordene los atributos de la clave de clasificación desde los más generales hasta los más específicos para maximizar la flexibilidad de las consultas.

Paso 5: uso de las condiciones de desigualdad en las claves de clasificación del índice secundario global

Las condiciones de desigualdad deben ser la última condición de la consulta. En este ejemplo, se muestra el uso de los operadores de comparación (`>=`, `BETWEEN`) y la coincidencia de prefijos (`begins_with()`) en los atributos de las claves de clasificación. Una vez que utilice un operador de desigualdad, no podrá agregar ninguna condición de clave de clasificación adicional después de este; la desigualdad debe ser la condición final de la expresión de condición de la clave.

Ejemplo de código

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, QueryCommand } from "@aws-sdk/lib-dynamodb";
```

```
const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Query 1: Round comparison (inequality on first sort key attribute)
console.log("Query 1: Matches from QUARTERFINALS onwards");
const query1 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 >= :round',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'QUARTERFINALS'
    }
}));
console.log(`  Found ${query1.Items.length} matches\n`);

// Query 2: Round range with BETWEEN
console.log("Query 2: Matches between QUARTERFINALS and SEMIFINALS");
const query2 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 BETWEEN :start AND :end',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':start': 'QUARTERFINALS',
        ':end': 'SEMIFINALS'
    }
}));
console.log(`  Found ${query2.Items.length} matches\n`);

// Query 3: Prefix matching with begins_with (treated as inequality)
console.log("Query 3: Matches in brackets starting with 'U'");
const query3 = await docClient.send(new QueryCommand({
```

```
    TableName: 'TournamentMatches',
    IndexName: 'TournamentRegionIndex',
    KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND round
 = :round AND begins_with(bracket, :prefix)',
    ExpressionAttributeNames: {
        '#region': 'region'  // 'region' is a reserved keyword
    },
    ExpressionAttributeValues: {
        ':tournament': 'WINTER2024',
        ':region': 'NA-EAST',
        ':round': 'SEMIFINALS',
        ':prefix': 'U'
    }
}));
console.log(`  Found ${query3.Items.length} matches\n`);

// Query 4: INVALID - condition after inequality
console.log("Query 4: Attempting condition after inequality (WILL FAIL)");
try {
    const query4 = await docClient.send(new QueryCommand({
        TableName: 'TournamentMatches',
        IndexName: 'TournamentRegionIndex',
        KeyConditionExpression: 'tournamentId = :tournament AND #region = :region AND
 round > :round AND bracket = :bracket',
        ExpressionAttributeNames: {
            '#region': 'region'  // 'region' is a reserved keyword
        },
        ExpressionAttributeValues: {
            ':tournament': 'WINTER2024',
            ':region': 'NA-EAST',
            ':round': 'QUARTERFINALS',
            ':bracket': 'UPPER'
        }
    }));
} catch (error) {
    console.log(`  Error: ${error.message}`);
    console.log(`  Cannot add conditions after inequality - it must be last\n`);
}
```

Reglas de operadores de desigualdad: puede usar operadores de comparación (>, >=, <, <=) BETWEEN para consultas de rango y `begins_with()` para hacer coincidir prefijos. La desigualdad deben ser la última condición de la consulta.

Patrones validos:

- Condiciones de igualdad seguidas de desigualdad: `round = 'SEMIFINALS' AND bracket = 'UPPER' AND matchId > 'match-001'`
- Desigualdad en el primer atributo: `round BETWEEN 'QUARTERFINALS' AND 'SEMIFINALS'`
- La coincidencia de prefijos como condición final: `round = 'SEMIFINALS' AND begins_with(bracket, 'U')`

Patrones no válidos:

- Agregación de condiciones después de una desigualdad: `round > 'QUARTERFINALS' AND bracket = 'UPPER'`
- Uso de múltiples desigualdades: `round > 'QUARTERFINALS' AND bracket > 'L'`

> **Important**
>
> `begins_with()` se trata como una condición de desigualdad, por lo que no puede seguirla ninguna condición de clave de clasificación adicional.

Paso 6: consulta del índice secundario global PlayerMatchHistoryIndex con la clave de clasificación de varios atributos

En este ejemplo, se consulta el PlayerMatchHistoryIndex, que tiene una clave de partición única (`player1Id`) y una clave de clasificación de varios atributos (`matchDate` + `round`). Esto permite analizar todos los torneos consultando todos los partidos de un jugador específico sin conocer los ID del torneo, mientras que la tabla base requeriría consultas independientes por combinación de torneo y región.

Ejemplo de código

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, QueryCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

// Query 1: All matches for Player 101 across all tournaments
```

```
console.log("Query 1: All matches for Player 101");
const query1 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'PlayerMatchHistoryIndex',
    KeyConditionExpression: 'player1Id = :player',
    ExpressionAttributeValues: {
        ':player': '101'
    }
}));

console.log(`  Found ${query1.Items.length} matches for Player 101:`);
query1.Items.forEach(match => {
    console.log(`    ${match.tournamentId}/${match.region} - ${match.matchDate} -
 ${match.round}`);
});
console.log();

// Query 2: Player 101 matches on specific date
console.log("Query 2: Player 101 matches on 2024-01-18");
const query2 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'PlayerMatchHistoryIndex',
    KeyConditionExpression: 'player1Id = :player AND matchDate = :date',
    ExpressionAttributeValues: {
        ':player': '101',
        ':date': '2024-01-18'
    }
}));

console.log(`  Found ${query2.Items.length} matches\n`);

// Query 3: Player 101 SEMIFINALS matches on specific date
console.log("Query 3: Player 101 SEMIFINALS matches on 2024-01-18");
const query3 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'PlayerMatchHistoryIndex',
    KeyConditionExpression: 'player1Id = :player AND matchDate = :date AND round
 = :round',
    ExpressionAttributeValues: {
        ':player': '101',
        ':date': '2024-01-18',
        ':round': 'SEMIFINALS'
    }
}));
```

```
console.log(`  Found ${query3.Items.length} matches\n`);

// Query 4: Player 101 matches in date range
console.log("Query 4: Player 101 matches in January 2024");
const query4 = await docClient.send(new QueryCommand({
    TableName: 'TournamentMatches',
    IndexName: 'PlayerMatchHistoryIndex',
    KeyConditionExpression: 'player1Id = :player AND matchDate BETWEEN :start
 AND :end',
    ExpressionAttributeValues: {
        ':player': '101',
        ':start': '2024-01-01',
        ':end': '2024-01-31'
    }
}));

console.log(`  Found ${query4.Items.length} matches\n`);
```

Variaciones de patrones

Datos de series temporales con claves de varios atributos

Optimización de las consultas de series temporales con atributos temporales jerárquicos

Ejemplo de código

```
{
    TableName: 'IoTReadings',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'readingId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'readingId', AttributeType: 'S' },
        { AttributeName: 'deviceId', AttributeType: 'S' },
        { AttributeName: 'locationId', AttributeType: 'S' },
        { AttributeName: 'year', AttributeType: 'S' },
        { AttributeName: 'month', AttributeType: 'S' },
        { AttributeName: 'day', AttributeType: 'S' },
        { AttributeName: 'timestamp', AttributeType: 'S' }
    ],
    // GSI with multi-attribute keys for time-series queries
```

```
    GlobalSecondaryIndexes: [{
        IndexName: 'DeviceLocationTimeIndex',
        KeySchema: [
            { AttributeName: 'deviceId', KeyType: 'HASH' },
            { AttributeName: 'locationId', KeyType: 'HASH' },
            { AttributeName: 'year', KeyType: 'RANGE' },
            { AttributeName: 'month', KeyType: 'RANGE' },
            { AttributeName: 'day', KeyType: 'RANGE' },
            { AttributeName: 'timestamp', KeyType: 'RANGE' }
        ],
        Projection: { ProjectionType: 'ALL' }
    }],
    BillingMode: 'PAY_PER_REQUEST'
}

// Query patterns enabled via GSI:
// - All readings for device in location
// - Readings for specific year
// - Readings for specific month in year
// - Readings for specific day
// - Readings in time range
```

Beneficios: la jerarquía temporal natural (año → mes → día → marca temporal) permite realizar consultas eficientes con una granularidad en cualquier momento sin necesidad de analizar ni manipular la fecha. El índice secundario global indexa automáticamente todas las lecturas con sus atributos de tiempo natural.

Pedidos de comercio electrónico con claves de varios atributos

Realización de un seguimiento de los pedidos con múltiples dimensiones

Ejemplo de código

```
{
    TableName: 'Orders',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'orderId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'orderId', AttributeType: 'S' },
        { AttributeName: 'sellerId', AttributeType: 'S' },
        { AttributeName: 'region', AttributeType: 'S' },
```

```
        { AttributeName: 'orderDate', AttributeType: 'S' },
        { AttributeName: 'category', AttributeType: 'S' },
        { AttributeName: 'customerId', AttributeType: 'S' },
        { AttributeName: 'orderStatus', AttributeType: 'S' }
    ],
    GlobalSecondaryIndexes: [
        {
            IndexName: 'SellerRegionIndex',
            KeySchema: [
                { AttributeName: 'sellerId', KeyType: 'HASH' },
                { AttributeName: 'region', KeyType: 'HASH' },
                { AttributeName: 'orderDate', KeyType: 'RANGE' },
                { AttributeName: 'category', KeyType: 'RANGE' },
                { AttributeName: 'orderId', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'CustomerOrdersIndex',
            KeySchema: [
                { AttributeName: 'customerId', KeyType: 'HASH' },
                { AttributeName: 'orderDate', KeyType: 'RANGE' },
                { AttributeName: 'orderStatus', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        }
    ],
    BillingMode: 'PAY_PER_REQUEST'
}

// SellerRegionIndex GSI queries:
// - Orders by seller and region
// - Orders by seller, region, and date
// - Orders by seller, region, date, and category

// CustomerOrdersIndex GSI queries:
// - Customer's orders
// - Customer's orders by date
// - Customer's orders by date and status
```

Datos de organización jerárquica

Jerarquías organizativas de modelos

Ejemplo de código

```
{
    TableName: 'Employees',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'employeeId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'employeeId', AttributeType: 'S' },
        { AttributeName: 'companyId', AttributeType: 'S' },
        { AttributeName: 'divisionId', AttributeType: 'S' },
        { AttributeName: 'departmentId', AttributeType: 'S' },
        { AttributeName: 'teamId', AttributeType: 'S' },
        { AttributeName: 'skillCategory', AttributeType: 'S' },
        { AttributeName: 'skillLevel', AttributeType: 'S' },
        { AttributeName: 'yearsExperience', AttributeType: 'N' }
    ],
    GlobalSecondaryIndexes: [
        {
            IndexName: 'OrganizationIndex',
            KeySchema: [
                { AttributeName: 'companyId', KeyType: 'HASH' },
                { AttributeName: 'divisionId', KeyType: 'HASH' },
                { AttributeName: 'departmentId', KeyType: 'RANGE' },
                { AttributeName: 'teamId', KeyType: 'RANGE' },
                { AttributeName: 'employeeId', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'SkillsIndex',
            KeySchema: [
                { AttributeName: 'skillCategory', KeyType: 'HASH' },
                { AttributeName: 'skillLevel', KeyType: 'RANGE' },
                { AttributeName: 'yearsExperience', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'INCLUDE', NonKeyAttributes: ['employeeId',
 'name'] }
        }
    ],
    BillingMode: 'PAY_PER_REQUEST'
}
```

```
// OrganizationIndex GSI query patterns:
// - All employees in company/division
// - Employees in specific department
// - Employees in specific team

// SkillsIndex GSI query patterns:
// - Employees by skill and experience level
```

Claves dispersas de varios atributos

Combinación de claves de varios atributos para crear un GSI disperso

Ejemplo de código

```
{
    TableName: 'Products',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'productId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'productId', AttributeType: 'S' },
        { AttributeName: 'categoryId', AttributeType: 'S' },
        { AttributeName: 'subcategoryId', AttributeType: 'S' },
        { AttributeName: 'averageRating', AttributeType: 'N' },
        { AttributeName: 'reviewCount', AttributeType: 'N' }
    ],
    GlobalSecondaryIndexes: [
        {
            IndexName: 'CategoryIndex',
            KeySchema: [
                { AttributeName: 'categoryId', KeyType: 'HASH' },
                { AttributeName: 'subcategoryId', KeyType: 'HASH' },
                { AttributeName: 'productId', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'ReviewedProductsIndex',
            KeySchema: [
                { AttributeName: 'categoryId', KeyType: 'HASH' },
                { AttributeName: 'averageRating', KeyType: 'RANGE' },  // Optional
 attribute
```

```
                { AttributeName: 'reviewCount', KeyType: 'RANGE' }     // Optional
 attribute
            ],
            Projection: { ProjectionType: 'ALL' }
        }
    ],
    BillingMode: 'PAY_PER_REQUEST'
}

// Only products with reviews appear in ReviewedProductsIndex GSI
// Automatic filtering without application logic
// Multi-attribute sort key enables rating and count queries
```

Multitenencia de SaaS

Plataforma SaaS multiusuario con aislamiento de clientes

Ejemplo de código

```
// Table design
{
    TableName: 'SaasData',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'resourceId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'resourceId', AttributeType: 'S' },
        { AttributeName: 'tenantId', AttributeType: 'S' },
        { AttributeName: 'customerId', AttributeType: 'S' },
        { AttributeName: 'resourceType', AttributeType: 'S' }
    ],
    // GSI with multi-attribute keys for tenant-customer isolation
    GlobalSecondaryIndexes: [{
        IndexName: 'TenantCustomerIndex',
        KeySchema: [
            { AttributeName: 'tenantId', KeyType: 'HASH' },
            { AttributeName: 'customerId', KeyType: 'HASH' },
            { AttributeName: 'resourceType', KeyType: 'RANGE' },
            { AttributeName: 'resourceId', KeyType: 'RANGE' }
        ],
        Projection: { ProjectionType: 'ALL' }
    }],
    BillingMode: 'PAY_PER_REQUEST'
```

```
}

// Query GSI: All resources for tenant T001, customer C001
const resources = await docClient.send(new QueryCommand({
    TableName: 'SaasData',
    IndexName: 'TenantCustomerIndex',
    KeyConditionExpression: 'tenantId = :tenant AND customerId = :customer',
    ExpressionAttributeValues: {
        ':tenant': 'T001',
        ':customer': 'C001'
    }
}));

// Query GSI: Specific resource type for tenant/customer
const documents = await docClient.send(new QueryCommand({
    TableName: 'SaasData',
    IndexName: 'TenantCustomerIndex',
    KeyConditionExpression: 'tenantId = :tenant AND customerId = :customer AND
 resourceType = :type',
    ExpressionAttributeValues: {
        ':tenant': 'T001',
        ':customer': 'C001',
        ':type': 'document'
    }
}));
```

Beneficios: consultas eficientes en el contexto de inquilino-cliente y organización natural de los datos.

Transacciones financieras

Sistema bancario que rastrea las transacciones de las cuentas mediante GSI

Ejemplo de código

```
// Table design
{
    TableName: 'BankTransactions',
    // Base table: Simple partition key
    KeySchema: [
        { AttributeName: 'transactionId', KeyType: 'HASH' }
    ],
    AttributeDefinitions: [
        { AttributeName: 'transactionId', AttributeType: 'S' },
        { AttributeName: 'accountId', AttributeType: 'S' },
```

```
        { AttributeName: 'year', AttributeType: 'S' },
        { AttributeName: 'month', AttributeType: 'S' },
        { AttributeName: 'day', AttributeType: 'S' },
        { AttributeName: 'transactionType', AttributeType: 'S' }
    ],
    GlobalSecondaryIndexes: [
        {
            IndexName: 'AccountTimeIndex',
            KeySchema: [
                { AttributeName: 'accountId', KeyType: 'HASH' },
                { AttributeName: 'year', KeyType: 'RANGE' },
                { AttributeName: 'month', KeyType: 'RANGE' },
                { AttributeName: 'day', KeyType: 'RANGE' },
                { AttributeName: 'transactionId', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        },
        {
            IndexName: 'TransactionTypeIndex',
            KeySchema: [
                { AttributeName: 'accountId', KeyType: 'HASH' },
                { AttributeName: 'transactionType', KeyType: 'RANGE' },
                { AttributeName: 'year', KeyType: 'RANGE' },
                { AttributeName: 'month', KeyType: 'RANGE' }
            ],
            Projection: { ProjectionType: 'ALL' }
        }
    ],
    BillingMode: 'PAY_PER_REQUEST'
}

// Query AccountTimeIndex GSI: All transactions for account in 2023
const yearTransactions = await docClient.send(new QueryCommand({
    TableName: 'BankTransactions',
    IndexName: 'AccountTimeIndex',
    KeyConditionExpression: 'accountId = :account AND #year = :year',
    ExpressionAttributeNames: { '#year': 'year' },
    ExpressionAttributeValues: {
        ':account': 'ACC-12345',
        ':year': '2023'
    }
}));

// Query AccountTimeIndex GSI: Transactions in specific month
```

```
const monthTransactions = await docClient.send(new QueryCommand({
    TableName: 'BankTransactions',
    IndexName: 'AccountTimeIndex',
    KeyConditionExpression: 'accountId = :account AND #year = :year AND #month
 = :month',
    ExpressionAttributeNames: { '#year': 'year', '#month': 'month' },
    ExpressionAttributeValues: {
        ':account': 'ACC-12345',
        ':year': '2023',
        ':month': '11'
    }
}));

// Query TransactionTypeIndex GSI: Deposits in 2023
const deposits = await docClient.send(new QueryCommand({
    TableName: 'BankTransactions',
    IndexName: 'TransactionTypeIndex',
    KeyConditionExpression: 'accountId = :account AND transactionType = :type AND #year
 = :year',
    ExpressionAttributeNames: { '#year': 'year' },
    ExpressionAttributeValues: {
        ':account': 'ACC-12345',
        ':type': 'deposit',
        ':year': '2023'
    }
}));
```

Ejemplo completo

El siguiente ejemplo muestra las claves de varios atributos desde la configuración hasta la limpieza:

Ejemplo de código

```
import {
    DynamoDBClient,
    CreateTableCommand,
    DeleteTableCommand,
    waitUntilTableExists
} from "@aws-sdk/client-dynamodb";
import {
    DynamoDBDocumentClient,
    PutCommand,
    QueryCommand
} from "@aws-sdk/lib-dynamodb";
```

```
const client = new DynamoDBClient({ region: 'us-west-2' });
const docClient = DynamoDBDocumentClient.from(client);

async function multiAttributeKeysDemo() {
    console.log("Starting Multi-Attribute GSI Keys Demo\n");

    // Step 1: Create table with GSIs using multi-attribute keys
    console.log("1. Creating table with multi-attribute GSI keys...");
    await client.send(new CreateTableCommand({
        TableName: 'TournamentMatches',
        KeySchema: [
            { AttributeName: 'matchId', KeyType: 'HASH' }
        ],
        AttributeDefinitions: [
            { AttributeName: 'matchId', AttributeType: 'S' },
            { AttributeName: 'tournamentId', AttributeType: 'S' },
            { AttributeName: 'region', AttributeType: 'S' },
            { AttributeName: 'round', AttributeType: 'S' },
            { AttributeName: 'bracket', AttributeType: 'S' },
            { AttributeName: 'player1Id', AttributeType: 'S' },
            { AttributeName: 'matchDate', AttributeType: 'S' }
        ],
        GlobalSecondaryIndexes: [
            {
                IndexName: 'TournamentRegionIndex',
                KeySchema: [
                    { AttributeName: 'tournamentId', KeyType: 'HASH' },
                    { AttributeName: 'region', KeyType: 'HASH' },
                    { AttributeName: 'round', KeyType: 'RANGE' },
                    { AttributeName: 'bracket', KeyType: 'RANGE' },
                    { AttributeName: 'matchId', KeyType: 'RANGE' }
                ],
                Projection: { ProjectionType: 'ALL' }
            },
            {
                IndexName: 'PlayerMatchHistoryIndex',
                KeySchema: [
                    { AttributeName: 'player1Id', KeyType: 'HASH' },
                    { AttributeName: 'matchDate', KeyType: 'RANGE' },
                    { AttributeName: 'round', KeyType: 'RANGE' }
                ],
                Projection: { ProjectionType: 'ALL' }
            }
```

```
        ],
        BillingMode: 'PAY_PER_REQUEST'
    }));

    await waitUntilTableExists({ client, maxWaitTime: 120 }, { TableName:
 'TournamentMatches' });
    console.log("Table created\n");

    // Step 2: Insert tournament matches
    console.log("2. Inserting tournament matches...");
    const matches = [
        { matchId: 'match-001', tournamentId: 'WINTER2024', region: 'NA-EAST', round:
 'FINALS', bracket: 'CHAMPIONSHIP', player1Id: '101', player2Id: '103', matchDate:
 '2024-01-20', winner: '101', score: '3-1' },
        { matchId: 'match-002', tournamentId: 'WINTER2024', region: 'NA-EAST', round:
 'SEMIFINALS', bracket: 'UPPER', player1Id: '101', player2Id: '105', matchDate:
 '2024-01-18', winner: '101', score: '3-2' },
        { matchId: 'match-003', tournamentId: 'WINTER2024', region: 'NA-WEST', round:
 'FINALS', bracket: 'CHAMPIONSHIP', player1Id: '102', player2Id: '104', matchDate:
 '2024-01-20', winner: '102', score: '3-2' },
        { matchId: 'match-004', tournamentId: 'SPRING2024', region: 'NA-EAST', round:
 'QUARTERFINALS', bracket: 'UPPER', player1Id: '101', player2Id: '108', matchDate:
 '2024-03-15', winner: '101', score: '3-0' }
    ];

    for (const match of matches) {
        await docClient.send(new PutCommand({ TableName: 'TournamentMatches', Item:
 match }));
    }
    console.log(`Inserted ${matches.length} tournament matches\n`);

    // Step 3: Query GSI with multi-attribute partition key
    console.log("3. Query TournamentRegionIndex GSI: WINTER2024/NA-EAST matches");
    const gsiQuery1 = await docClient.send(new QueryCommand({
        TableName: 'TournamentMatches',
        IndexName: 'TournamentRegionIndex',
        KeyConditionExpression: 'tournamentId = :tournament AND #region = :region',
        ExpressionAttributeNames: { '#region': 'region' },
        ExpressionAttributeValues: { ':tournament': 'WINTER2024', ':region': 'NA-
EAST' }
    }));

    console.log(`  Found ${gsiQuery1.Items.length} matches:`);
    gsiQuery1.Items.forEach(match => {
```

```
        console.log(`    ${match.round} - ${match.bracket} - ${match.winner} won`);
    });

    // Step 4: Query GSI with multi-attribute sort key
    console.log("\n4. Query PlayerMatchHistoryIndex GSI: All matches for Player 101");
    const gsiQuery2 = await docClient.send(new QueryCommand({
        TableName: 'TournamentMatches',
        IndexName: 'PlayerMatchHistoryIndex',
        KeyConditionExpression: 'player1Id = :player',
        ExpressionAttributeValues: { ':player': '101' }
    }));

    console.log(`  Found ${gsiQuery2.Items.length} matches for Player 101:`);
    gsiQuery2.Items.forEach(match => {
        console.log(`    ${match.tournamentId}/${match.region} - ${match.matchDate} -
 ${match.round}`);
    });

    console.log("\nDemo complete");
    console.log("No synthetic keys needed - GSIs use native attributes automatically");
}

async function cleanup() {
    console.log("Deleting table...");
    await client.send(new DeleteTableCommand({ TableName: 'TournamentMatches' }));
    console.log("Table deleted");
}

// Run demo
multiAttributeKeysDemo().catch(console.error);

// Uncomment to cleanup:
// cleanup().catch(console.error);
```

Estructura de código mínima

Ejemplo de código

```
// 1. Create table with GSI using multi-attribute keys
await client.send(new CreateTableCommand({
    TableName: 'MyTable',
    KeySchema: [
        { AttributeName: 'id', KeyType: 'HASH' }        // Simple base table PK
    ],
```

```
    AttributeDefinitions: [
        { AttributeName: 'id', AttributeType: 'S' },
        { AttributeName: 'attr1', AttributeType: 'S' },
        { AttributeName: 'attr2', AttributeType: 'S' },
        { AttributeName: 'attr3', AttributeType: 'S' },
        { AttributeName: 'attr4', AttributeType: 'S' }
    ],
    GlobalSecondaryIndexes: [{
        IndexName: 'MyGSI',
        KeySchema: [
            { AttributeName: 'attr1', KeyType: 'HASH' },    // GSI PK attribute 1
            { AttributeName: 'attr2', KeyType: 'HASH' },    // GSI PK attribute 2
            { AttributeName: 'attr3', KeyType: 'RANGE' },   // GSI SK attribute 1
            { AttributeName: 'attr4', KeyType: 'RANGE' }    // GSI SK attribute 2
        ],
        Projection: { ProjectionType: 'ALL' }
    }],
    BillingMode: 'PAY_PER_REQUEST'
}));

// 2. Insert items with native attributes (no concatenation needed for GSI)
await docClient.send(new PutCommand({
    TableName: 'MyTable',
    Item: {
        id: 'item-001',
        attr1: 'value1',
        attr2: 'value2',
        attr3: 'value3',
        attr4: 'value4',
        // ... other attributes
    }
}));

// 3. Query GSI with all partition key attributes
await docClient.send(new QueryCommand({
    TableName: 'MyTable',
    IndexName: 'MyGSI',
    KeyConditionExpression: 'attr1 = :v1 AND attr2 = :v2',
    ExpressionAttributeValues: {
        ':v1': 'value1',
        ':v2': 'value2'
    }
}));
```

```
// 4. Query GSI with sort key attributes (left-to-right)
await docClient.send(new QueryCommand({
    TableName: 'MyTable',
    IndexName: 'MyGSI',
    KeyConditionExpression: 'attr1 = :v1 AND attr2 = :v2 AND attr3 = :v3',
    ExpressionAttributeValues: {
        ':v1': 'value1',
        ':v2': 'value2',
        ':v3': 'value3'
    }
}));

// Note: If any attribute name is a DynamoDB reserved keyword, use
 ExpressionAttributeNames:
// KeyConditionExpression: 'attr1 = :v1 AND #attr2 = :v2'
// ExpressionAttributeNames: { '#attr2': 'attr2' }
```

Recursos adicionales

- Prácticas recomendadas de DynamoDB
- Uso de tablas y datos
- Índices secundarios globales
- Operaciones de consulta y análisis

Administración de índices secundarios globales en DynamoDB

En esta sección se describe cómo crear, modificar y eliminar índices secundarios globales en Amazon DynamoDB.

Temas

- Creación de una tabla con índices secundarios globales
- Descripción de los índices secundarios globales en una tabla
- Adición de un índice secundario global a una tabla existente
- Eliminación de un índice secundario global
- Modificación de un índice secundario global durante la creación

Creación de una tabla con índices secundarios globales

Para crear una tabla con uno o varios índices secundarios globales, use la operación `CreateTable` con el parámetro `GlobalSecondaryIndexes`. Para disfrutar de la máxima flexibilidad en las consultas, puede crear hasta 20 índices secundarios globales (cuota predeterminada) por tabla.

Debe especificar un atributo que actúe como clave de partición del índice. También tiene la opción de especificar otro atributo para la clave de ordenación del índice. No es necesario que ninguno de estos atributos de clave sea igual que un atributo de clave de la tabla. Por ejemplo, en la tabla GameScores (consulte Uso de índices secundarios globales en DynamoDB), ni `TopScore` ni `TopScoreDateTime` son atributos de clave. Puede crear un índice secundario global que tenga una clave de partición de `TopScore` y una clave de ordenación de `TopScoreDateTime`. Se podría usar un índice de este tipo para determinar si existe una correlación entre las mejores puntuaciones y la hora del día a la que se juega.

Todos los atributos de clave del índice deben ser escalares y pueden ser de tipo `String`, `Number` o `Binary`. No pueden ser documentos ni conjuntos. Puede proyectar atributos de cualquier tipo de datos en un índice secundario global. Esto incluye escalares, documentos y conjuntos. Para obtener una lista completa de los tipos de datos, consulte Tipos de datos.

Si usa el modo aprovisionado, debe proporcionar para el índice los ajustes de `ProvisionedThroughput`, que constan de `ReadCapacityUnits` y `WriteCapacityUnits`. Estos ajustes de rendimiento aprovisionado son independientes de los de la tabla pero se comportan de forma parecida. Para obtener más información, consulte Consideraciones sobre el rendimiento aprovisionado para los índices secundarios globales.

Los índices secundarios globales heredan el modo de capacidad de lectura/escritura de la tabla base. Para obtener más información, consulte Aspectos a tener en cuenta al cambiar los modos de capacidad en DynamoDB.

> **Note**
>
> Las operaciones de reposición y de escritura en curso comparten el rendimiento de escritura dentro del índice secundario global. Al crear un nuevo índice secundario global, resulta importante verificar si la elección de clave de partición produce una distribución desigual o restringida de datos o tráfico entre los valores de clave de partición del nuevo índice. Si esto ocurre, podría ver operaciones de reposición y escritura que se producen al mismo tiempo y que se limiten las escrituras en la tabla base. El servicio adopta medidas para minimizar el potencial de este escenario, pero no tiene información de la forma de los datos del cliente

con respecto a la clave de partición de índice, la proyección elegida o la escasez de la clave principal del índice.
Si sospecha que el nuevo índice secundario global puede tener datos estrechos o sesgados, o bien una distribución del tráfico entre los valores clave de partición, tenga en cuenta lo siguiente antes de agregar nuevos índices a tablas importantes desde el punto de vista operativo.

- Es posible que sea más seguro agregar el índice en un momento en que la aplicación esté generando la menor cantidad de tráfico.
- Considere la posibilidad de habilitar CloudWatch Contributor Insights en la tabla base y los índices. Esto le proporcionará información valiosa sobre la distribución del tráfico.
- Para tablas base del modo de capacidad aprovisionada e índices, establezca la capacidad de escritura aprovisionada del nuevo índice en al menos el doble de la de la tabla base. Vea las métricas de `WriteThrottleEvents`, `ThrottledRequests`, `OnlineIndexPercentageProgress`, `OnlineIndexConsumedWriteCapacity` y `OnlineIndexThrottleEvents` de CloudWatch durante todo el proceso. Ajuste la capacidad de escritura aprovisionada según sea necesario para completar la reposición en un tiempo razonable sin ningún efecto de limitación significativo en las operaciones en curso.
- Prepárese para cancelar la creación del índice si experimenta un impacto operativo debido a la limitación de escritura y el aumento de la capacidad de escritura aprovisionada en el nuevo índice secundario global no lo resuelve.

Descripción de los índices secundarios globales en una tabla

Para ver el estado de todos los índices secundarios globales de una tabla, se usa la operación `DescribeTable`. La parte `GlobalSecondaryIndexes` de la respuesta enumera todos los índices de la tabla, junto con el estado actual de cada uno de ellos (`IndexStatus`).

El `IndexStatus` para un índice secundario global será uno de los siguientes:

- `CREATING`: el índice está en proceso de creación y aún no está disponible para usarlo.
- `ACTIVE`: el índice está listo para usarlo y las aplicaciones pueden realizar operaciones `Query` en él.
- `UPDATING`: se están cambiando los ajustes de desempeño provisionado del índice.
- `DELETING`: el índice se está eliminando y ya no se puede utilizar.

Cuando DynamoDB ha terminado de crear un índice secundario global, el estado de este último cambia de `CREATING` a `ACTIVE`.

Adición de un índice secundario global a una tabla existente

Para agregar un índice secundario global a una tabla existente, se usa la operación `UpdateTable` con el parámetro `GlobalSecondaryIndexUpdates`. Debe proporcionar lo siguiente:

- El nombre del índice. Este nombre debe ser único entre todos los índices de la tabla.
- El esquema de claves del índice. Debe especificar un atributo para la clave de partición del índice y, si lo desea, otro atributo para la clave de ordenación del índice. No es necesario que ninguno de estos atributos de clave sea igual que un atributo de clave de la tabla. Los tipos de datos de cada atributo del esquema deben ser escalares: `String`, `Number` o `Binary`.
- Los atributos de la tabla que se van a proyectar en el índice:
  - `KEYS_ONLY`: cada elemento del índice consta únicamente de los valores de la clave de partición y la clave de ordenación de la tabla, así como de los valores de las claves del índice.
  - `INCLUDE`: además de los atributos que se describen en `KEYS_ONLY`, el índice secundario incluye otros atributos sin clave que se especifiquen.
  - `ALL`: el índice incluye todos los atributos de la tabla de origen.
- Los ajustes de rendimiento aprovisionado del índice, que constan de `ReadCapacityUnits` y `WriteCapacityUnits`. Estos ajustes de rendimiento aprovisionado son independientes de los de la tabla.

Solo puede crear un índice secundario global por operación `UpdateTable`.

> **Important**
>
> Al crear un índice secundario global (GSI), las operaciones de escritura en la tabla base se pueden limitar si la actividad del GSI resultante de las escrituras en la tabla base supera la capacidad de escritura aprovisionada del GSI. Esta limitación afecta a todas las operaciones de escritura, desde el proceso de indexación hasta la posible interrupción de las cargas de trabajo de producción. Para obtener más información, consulte Solución de problemas de limitación en Amazon DynamoDB.

Fases de creación del índice

Cuando se agrega un nuevo índice secundario global a una tabla existente, la tabla sigue estando disponible mientras se crea el índice. Sin embargo, el nuevo índice no estará disponible para las operaciones de consulta hasta que el estado cambie de `CREATING` a `ACTIVE`.

> **Note**
>
> La creación de índices secundarios globales no utiliza Application Auto Scaling. El aumento de la capacidad de `MIN` de Application Auto Scaling no disminuirá el tiempo de creación del índice secundario global.

En segundo plano, DynamoDB crea el índice en dos fases:

Asignación de recursos

DynamoDB asigna los recursos informáticos y de almacenamiento que se necesitan para crear el índice.

Durante la fase de asignación de recursos, el valor del atributo `IndexStatus` es `CREATING` y el valor del atributo `Backfilling` es false. Use la operación `DescribeTable` para recuperar el estado de una tabla y todos sus índices secundarios.

Mientras el índice se encuentra en la fase de asignación de recursos, no se puede eliminar el índice ni eliminar su tabla principal. Tampoco se puede modificar el rendimiento aprovisionado del índice ni de la tabla. No se pueden agregar ni eliminar otros índices de la tabla. Sin embargo, se puede modificar el rendimiento aprovisionado de estos otros índices.

Reposición

Para cada elemento de la tabla, DynamoDB determina el conjunto de atributos que se escribirá en el índice según su proyección (`KEYS_ONLY`, `INCLUDE` o `ALL`). A continuación, escribe estos atributos en el índice. Durante la fase de reposición, DynamoDB lleva un seguimiento de los elementos que se agregan, eliminan o actualizan en la tabla. Los atributos de estos elementos también se agregan, eliminan o actualizan en el índice, según proceda.

Durante la fase de reposición, el atributo `IndexStatus` está establecido en `CREATING` y el atributo `Backfilling` está establecido en true. Use la operación `DescribeTable` para recuperar el estado de una tabla y todos sus índices secundarios.

Mientras el índice se encuentra en la fase de reposición, no se puede eliminar su tabla principal. Sin embargo, sí se puede eliminar el índice o modificar el rendimiento aprovisionado de la tabla y de cualquiera de sus índices secundarios globales.

> **Note**
>
> Durante la fase de reposición, algunas escrituras de elementos infractores podrían realizarse correctamente y otras se rechazarán. Después de la reposición, se rechazarán todas las escrituras en elementos que infrinjan el esquema de claves del nuevo índice. Recomendamos ejecutar la herramienta Violation Detector una vez que haya finalizado la fase de reposición para detectar y resolver cualquier infracción de clave que pueda haberse producido. Para obtener más información, consulte Detección y corrección de infracciones de la clave del índice en DynamoDB.

Mientras las fases de asignación de recursos y reposición están en curso, el índice se encuentra en el estado `CREATING`. Durante este plazo de tiempo, DynamoDB realiza operaciones de lectura en al tabla. No se le cobrará por las operaciones de lectura de la tabla base para rellenar el índice secundario global. Sin embargo, se le cobran las operaciones de escritura para rellenar el índice secundario global recién creado.

Una vez que se completa la creación del índice, su estado cambia a `ACTIVE`. No puede `Query` ni `Scan` el índice hasta que esté `ACTIVE`.

> **Note**
>
> En algunos casos, DynamoDB no puede escribir datos de la tabla en el índice porque se producen infracciones en las claves del índice. Esto puede ocurrir si:
>
> - El tipo de dato de un valor de atributo no coincide con el del esquema de claves de índice.
> - El tamaño de un atributo supera la longitud máxima de un atributo de clave del índice.
> - Un atributo de clave del índice tiene un valor binario o de cadena vacío.
>
> Las infracciones de claves del índice no interfieren con la creación de un índice secundario global. Sin embargo, cuando el índice adquiera el estado `ACTIVE`, las claves infractoras no estarán presentes en él.

DynamoDB proporciona una herramienta independiente para encontrar y resolver estos problemas. Para obtener más información, consulte Detección y corrección de infracciones de la clave del índice en DynamoDB.

Adición de un índice secundario global a una tabla grande

El tiempo necesario para crear un índice secundario global depende de varios factores, como los siguientes:

- El tamaño de la tabla.
- El número de elementos de la tabla que son aptos para incluirlos en el índice.
- El número de atributos que se proyectan en el índice.
- La capacidad de escritura aprovisionada del índice.
- La actividad de escritura en la tabla principal mientras se crean los índices.

Si se va a agregar un índice secundario global a una tabla muy grande, el proceso de creación puede tardar mucho tiempo en completarse. Para monitorear el progreso y determinar si el índice tiene suficiente capacidad de escritura, consulte las siguientes métricas de Amazon CloudWatch:

- `OnlineIndexPercentageProgress`
- `OnlineIndexConsumedWriteCapacity`
- `OnlineIndexThrottleEvents`

Para obtener más información acerca de cómo las métricas de CloudWatch se relacionan con DynamoDB, consulte Métricas de DynamoDB.

Important

Es posible que tenga que permitir listas de tablas muy grandes antes de crear o actualizar un índice secundario global. Póngase en contacto con AWS Support para incluir las tablas en la lista de permitidos.

> **Note**
>
> Por regla general, recomendamos establecer la capacidad de escritura aprovisionada del índice en 1,5 veces la capacidad de escritura de la tabla. Este ajuste es adecuado en muchos casos de uso. No obstante, puede que sus requisitos reales sean superiores o inferiores.

Mientras el índice está en la fase de reposición, DynamoDB utiliza la capacidad interna del sistema para leer la tabla. Esto permite minimizar el impacto de la creación del índice y asegurarse de que no se agote la capacidad de lectura de la tabla.

Sin embargo, es posible que el volumen de una actividad de escritura entrante supere la capacidad de lectura aprovisionada del índice. En este caso, se produciría un cuello de botella y la creación del índice tardaría más porque la actividad de escritura en el índice sería objeto de una limitación controlada. Mientras se crea el índice, recomendamos monitorear las métricas de Amazon CloudWatch del índice para determinar si su capacidad de escritura consumida es superior a la aprovisionada. Si se produce un cuello de botella, debe aumentar la capacidad de escritura aprovisionada en el índice para evitar que se imponga la limitación controlada a las escrituras durante la fase de reposición.

Una vez que se haya creado el índice, conviene establecer su capacidad de escritura aprovisionada de modo que refleje el uso normal de la aplicación.

Eliminación de un índice secundario global

Si ya no necesita un índice secundario global, puede eliminarlo mediante la ooperación `UpdateTable`.

Solo se puede eliminar un índice secundario global por operación `UpdateTable`.

Mientras se elimina el índice secundario global, no afecta en absoluto a la actividad de lectura o escritura en la tabla principal. Mientras la eliminación está en curso, puede modificar el rendimiento aprovisionado de los demás índices.

> **Note**
>
> - Al eliminar una tabla mediante la acción `DeleteTable`, todos los índices secundarios globales de esa tabla se eliminan también.

- No se cargará en su cuenta la operación de eliminación del índice secundario global.

Modificación de un índice secundario global durante la creación

Mientras se está creando un índice, puede usar la operación `DescribeTable` para determinar en qué fase se encuentra. La descripción del índice incluye un atributo de tipo Boolean, `Backfilling`, que indica si DynamoDB está cargando elementos de la tabla en el índice en un momento dado. Si `Backfilling` es true, significa que la fase de asignación de recursos se ha completado y el índice se encuentra en la fase de reposición.

Mientras que la reposición está en curso, puede actualizar los parámetros de rendimiento aprovisionado del índice. Puede optar por esta opción para agilizar la creación del índice: puede aumentar la capacidad de escritura del índice mientras se crea y reducirla después. Para modificar los ajustes de rendimiento aprovisionado del índice, utilice la operación `UpdateTable`. El estado del índice cambia a `UPDATING` y el valor de `Backfilling` es true hasta que el índice esté listo para usarlo.

Durante la fase de reposición, puede eliminar el índice que se está creando. Durante esta fase, no se pueden agregar ni eliminar otros índices de la tabla.

> **Note**
>
> En el caso de los índices que se han creado como parte de una operación `CreateTable`, el atributo `Backfilling` no aparece en el resultado de `DescribeTable`. Para obtener más información, consulte Fases de creación del índice.

Detección y corrección de infracciones de la clave del índice en DynamoDB

Durante la fase de reposición de la creación de un índice secundario global, Amazon DynamoDB examina cada elemento de la tabla para determinar si es apto para incluirlo en el índice. Algunos elementos podrían no ser aptos por la posibilidad de que causen infracciones de claves de índice. En estos casos, el elemento en cuestión permanecen en la tabla, pero el índice no incluye la entrada correspondiente a dicho elemento.

Un registro infracción de clave de índice se produce en las siguientes situaciones:

- Los tipos de datos del valor de un atributo y del esquema de claves del índice no coinciden. Por ejemplo, supongamos que uno de los elementos de la tabla `GameScores` tiene un atributo `TopScore` cuyo valor es de tipo `String`. Si agrega un índice secundario global con una clave de partición de `TopScore` de tipo `Number`, el elemento de la tabla infringiría la clave de índice.
- El valor de atributo de la tabla supera la longitud máxima de un atributo de clave del índice. La longitud máxima de una clave de partición es de 2048 bytes y la longitud máxima de una clave de ordenación es de 1024 bytes. Si cualquiera de los valores de atributos correspondientes de la tabla supera estos límites, el elemento de la tabla infringiría la clave de índice.

> **Note**
>
> Si se establece un valor binario o de cadena en un atributo que se utiliza como clave de índice, este valor debe tener una longitud mayor que cero; de lo contrario, el elemento de la tabla podría infringir la clave del índice.
> En la actualidad, esta herramienta no indica esta infracción de clave de índice de ninguna forma.

Si se produce una infracción de índice, la fase de reposición continúa sin interrupción. Sin embargo, los elementos infractores no se incluyen en el índice. Una vez que haya finalizado la fase de reposición, se rechazarán todas las escrituras en elementos que infringen el esquema de claves del nuevo índice.

Para identificar y corregir los valores de los atributos de una tabla que infringen una clave de índice, se utiliza la herramienta Violation Detector. Para ejecutar Violation Detector, debe crear un archivo de configuración que especifique el nombre de la tabla que se va a examinar, los nombres y los tipos de datos de las claves de partición y de ordenación del índice secundario global y qué medidas deben adoptarse si se detecta cualquier infracción de clave de índice. Violation Detector puede ejecutarse en dos modos diferentes:

- Modo de detección: detecta las infracciones de claves de índice. El modo de detección se utiliza para registrar los elementos de la tabla que causarían infracciones de claves en un índice secundario global. (Si lo desea, puede solicitar que estos elementos infractores de la tabla se eliminen de inmediato en cuanto se encuentren). El resultado del modo de detección se escribe en un archivo, que se puede utilizar para analizar los datos posteriormente.
- Modo de corrección: corrige las infracciones de claves de índice. En el modo de corrección, Violation Detector lee un archivo de información de entrada con el mismo formato que el archivo

de resultados del modo de detección. En el modo de corrección se leen los registros del archivo de información de entrada y, para cada registro, se elimina o actualiza el elemento correspondiente de la tabla. Tenga en cuenta que si decide actualizar los elementos, deberá editar el archivo de información de entrada y establecer valores adecuados para estas actualizaciones.

Descargar y ejecutar Violation Detector

Violation Detector está disponible como archivo Java ejecutable (`.jar`) y se ejecuta en ordenadores Windows, Mac o Linux. Violation Detector requiere Java 1.7 (o posterior) y Apache Maven.

- Descargar Violation Detector de GitHub

Siga las instrucciones del archivo `README.md` para descargar e instalar Violation Detector mediante Maven.

Para iniciar Violation Detector, vaya al directorio donde ha construido `ViolationDetector.java` e ingrese el siguiente comando.

```
java -jar ViolationDetector.jar [options]
```

La línea de comandos de Violation Detector acepta las siguientes opciones:

- `-h | --help`: imprime un resumen de uso y las opciones posibles de Violation Detector.
- `-p | --configFilePath value`: nombre completo de un archivo de configuración de Violation Detector. Para obtener más información, consulte El archivo de configuración de Violation Detector.
- `-t | --detect value`: detecta las infracciones de claves de índice de la tabla y las escribe en el archivo de resultados de Violation Detector. Si el valor de este parámetro se establece en `keep`, los elementos con infracciones de claves no se modifican. Si el valor se establece en `delete`, los elementos con infracciones de claves se eliminan de la tabla.
- `-c | --correct value`: lee las infracciones de claves de índice en un archivo de información de entrada y adopta medidas para corregir los elementos de la tabla. Si el valor de este parámetro se establece en `update`, los elementos con infracciones de claves se actualizan a otros valores sin infracciones. Si el valor se establece en `delete`, los elementos con infracciones de claves se eliminan de la tabla.

El archivo de configuración de Violation Detector

En tiempo de ejecución, la herramienta Violation Detector requiere un archivo de configuración. Los parámetros de este archivo determinan a qué recursos de DynamoDB puede obtener acceso Violation Detector y cuánto rendimiento aprovisionado puede consumir. En la siguiente tabla se describen estos parámetros.

Nombre del parámetro	Descripción	¿Obligatorio?

`awsCredentialsFile`	Nombre completo de un archivo que contiene las credenciales de AWS. El archivo de credenciales debe tener el siguiente formato:
	`accessKey = access_key_id_goes_here`
	`secretKey = secret_key_goes_here`
		Sí

`dynamoDBRegion`	Región de AWS en la que la reside la tabla. Por ejemplo: `us-west-2` .	Sí
`tableName`	Nombre de la tabla de DynamoDB que se va a examinar.	Sí
`gsiHashKeyName`	Nombre de la clave de partición del índice.	Sí
`gsiHashKeyType`	Tipo de datos de la clave de partición del índice: `String`, `Number` o `Binary`: `S | N | B`	Sí
`gsiRangeKeyName`	Nombre de la clave de ordenación del índice. No	No

Nombre del parámetro	Descripción	¿Obligatorio?
	especifique este parámetro si el índice solo tiene una clave principal simple (clave de partición).	
`gsiRangeKeyType`	Tipo de datos de la clave de ordenación del índice: `String`, `Number` o `Binary`: `S | N | B` No especifique este parámetro si el índice solo tiene una clave principal simple (clave de partición).	No
`recordDetails`	Indica si se deben escribir todos los detalles de las infracciones de claves de índice en el archivo de resultados. Si se establece en `true` (valor predeterminado), se registra toda la información sobre los elementos infractor es. Si se establece en `false`, solo se registra el número de infracciones.	No

Nombre del parámetro	Descripción	¿Obligatorio?
`recordGsiValueInViolationRecord`	Indica si se deben escribir los valores de las claves de índice infractoras en el archivo de resultados. Si se establece en `true` (predeterminado), se registran los valores de las claves. Si se establece en `false`, los valores de las claves no se registran.	No

Nombre del parámetro	Descripción	¿Obligatorio?

`detectionOutputPath`	Ruta completa del archivo de resultados de Violation Detector;. Este parámetro es compatible con la escritura en un directorio local o en Amazon Simple Storage Service (Amazon S3). A continuación se muestran algunos ejemplos:
	
	`detectionOutputPath = //local/path/filename.csv`
	
	`detectionOutputPath = s3://bucket/filename.csv`
	
	La información del archivo de resultados aparece en formato CSV (valores separados por comas). Si no se establece `detectionOutputPath` , el archivo de resultados se denomina `violation_detection.csv` y se escribe en el directorio de trabajo actual.
		No

Nombre del parámetro	Descripción	¿Obligatorio?
`numOfSegments`	El número de segmentos de examen en paralelo que se utilizarán cuando Violation Detector analice la tabla. El valor predeterminado es 1, lo que significa que la tabla se examina de forma secuencial. Si el valor es 2 o superior, entonces Violation Detector dividirá la tabla en esa cantidad de segmentos lógicos y un número igual de subprocesos de examen. El ajuste máximo de `numOfSegments` es 4096. En el caso de las tablas de mayor tamaño, un examen en paralelo suele resultar más rápido que un examen secuencial. Además, si la tabla es lo bastante grande para abarcar varias particiones, un examen en paralelo distribuye su actividad de lectura de manera uniforme entre varias particiones. Para obtener más información sobre los análisis en paralelo en DynamoDB, consulte Análisis paralelo.	No

Nombre del parámetro	Descripción	¿Obligatorio?
`numOfViolations`	Límite superior de infracciones de claves de índice que se escribirán en el archivo de resultados. Si se establece en `-1` (valor predeterminado), se examina toda la tabla. Si se establece en un número entero positivo, entonces Violation Detector se detendrá después de haber detectado esa cantidad de infracciones.	No
`numOfRecords`	Número de elementos de la tabla que se va a examinar. Si se establece en -1 (valor predeterminado), se examinará toda la tabla. Si se establece en un número entero positivo, entonces Violation Detector se detendrá después de haber examinado esa cantidad de elementos en la tabla.	No

Nombre del parámetro	Descripción	¿Obligatorio?
`readWriteIOPSPercent`	Regula el porcentaje de unidades de capacidad de lectura provisionadas que se consumen durante un examen de la tabla. Los valores válidos van de 1 a `100`. El valor predeterminado (`25`) significa que Violation Detector consumirá no más del 25 % del rendimiento de lectura aprovisionado de la tabla.	No
`correctionInputPath`	Ruta completa del archivo de información de entrada de corrección de Violation Detector. Si ejecuta Violation Detector en modo de corrección, el contenido de este archivo se usa para modificar o eliminar los elementos de datos de la tabla que infringen el índice secundario global. El formato del archivo `correctionInputPath` es el mismo que el del archivo `detectionOutputPath`. Esto permite procesar el resultado del modo de detección como información de entrada en el modo de corrección.	No

Nombre del parámetro	Descripción	¿Obligatorio?

`correctionOutputPath`	Ruta completa del archivo de resultados de corrección de Violation Detector. Este archivo se crea solo si existen errores de actualización.
	
	Este parámetro es compatible con la escritura en un directorio local o en Amazon S3. A continuación se muestran algunos ejemplos:
	
	`correctionOutputPath = //local/path/filename.csv`
	
	`correctionOutputPath = s3://bucket/filename.csv`
	
	La información del archivo de resultados aparece en formato CSV. Si no se establece `correctionOutputPath`, el archivo de resultados se denomina `violation_update_errors.csv` y se escribe en el directorio de trabajo actual.
		No

Detección

Para detectar infracciones de claves de índice, use Violation Detector con la opción de línea de comandos `--detect`. Para ver cómo funciona esta opción, tomemos la tabla `ProductCatalog`. A

continuación se muestra una lista de elementos de la tabla. Solo se muestran la clave principal (`Id`) y el atributo `Price`.

Id (clave principal)	Precio
101	5
102	20
103	200
201	100
202	200
203	300
204	400
205	500

Todos los valores de `Price` son de tipo `Number`. No obstante, dado que en DynamoDB no se usan esquemas, es posible agregar un elemento con un valor de `Price` no numérico. Por ejemplo, supongamos que agrega otro elemento a la tabla `ProductCatalog`.

Id (clave principal)	Precio
999	"Hello"

Ahora, la tabla contiene un total de nueve elementos.

Ahora agrega un nuevo índice secundario global a la tabla: `PriceIndex`. La clave principal de este índice es una clave de partición, `Price`, que es de tipo `Number`. Después de crearlo, el índice contendrá ocho elementos. Sin embargo, la tabla `ProductCatalog` tiene nueve elementos. El motivo de esta discrepancia es que, aunque el valor `"Hello"` es de tipo `String`, la clave principal de `PriceIndex` es de tipo `Number`. El valor `String` infringe la clave del índice secundario global, por lo que no está presente en el índice.

Para usar Violation Detector en esta situación, primero se crea un archivo de configuración, como el siguiente.

```
# Properties file for violation detection tool configuration.
# Parameters that are not specified will use default values.

awsCredentialsFile = /home/alice/credentials.txt
dynamoDBRegion = us-west-2
tableName = ProductCatalog
gsiHashKeyName = Price
gsiHashKeyType = N
recordDetails = true
recordGsiValueInViolationRecord = true
detectionOutputPath = ./gsi_violation_check.csv
correctionInputPath = ./gsi_violation_check.csv
numOfSegments = 1
readWriteIOPSPercent = 40
```

A continuación, se ejecuta el Violation Detector como en el siguiente ejemplo.

```
$  java -jar ViolationDetector.jar --configFilePath config.txt --detect keep

Violation detection started: sequential scan, Table name: ProductCatalog, GSI name:
 PriceIndex
Progress: Items scanned in total: 9,    Items scanned by this thread: 9,    Violations
 found by this thread: 1, Violations deleted by this thread: 0
Violation detection finished: Records scanned: 9, Violations found: 1, Violations
 deleted: 0, see results at: ./gsi_violation_check.csv
```

Si el parámetro de configuración `recordDetails` se establece en `true`, entonces Violation Detector escribe los detalles de cada infracción en el archivo de resultados, como en el siguiente ejemplo.

```
Table Hash Key,GSI Hash Key Value,GSI Hash Key Violation Type,GSI Hash Key Violation
 Description,GSI Hash Key Update Value(FOR USER),Delete Blank Attributes When Updating?
(Y/N)

999,"{""S"":""Hello""}",Type Violation,Expected: N Found: S,,
```

El archivo de salida está en formato CSV. La primera línea del archivo es un encabezado, seguido de un registro por cada elemento que infringe la clave del índice. Los campos de estos registros de infracción son los siguientes:

- Table hash key: valor de la clave de partición del elemento de la tabla.
- Table range key: valor de la clave de clasificación del elemento de la tabla.
- GSI hash key value: valor de la clave de partición del índice secundario global.
- GSI hash key violation type: puede ser `Type Violation` o `Size Violation`.
- GSI hash key violation description: causa de la infracción.
- GSI hash key update value (FOR USER): en el modo de corrección, nuevo valor suministrado por el usuario para el atributo.
- GSI range key values: el valor de la clave de clasificación del índice secundario global.
- GSI range key violation type: puede ser `Type Violation` o `Size Violation`.
- GSI range key violation description: causa de la infracción.
- GSI range key update value (FOR USER): en el modo de corrección, nuevo valor suministrado por el usuario para el atributo.
- Delete blank attribute when updating (Y/N): en el modo de corrección, determina si el elemento infractor se eliminará (Y) o se conservará (N) en la tabla, pero solamente si cualquiera de los campos siguientes está en blanco:
  - `GSI Hash Key Update Value(FOR USER)`
  - `GSI Range Key Update Value(FOR USER)`

  Si alguno de estos campos no está en blanco, entonces `Delete Blank Attribute When Updating(Y/N)` no surte ningún efecto.

> **Note**
>
> El formato de los resultados puede variar, en función del archivo de configuración y de las opciones de la línea de comandos. Por ejemplo, si la tabla tiene una clave principal simple (sin clave de ordenación), no habrá ningún campo de clave de ordenación en el resultado. Los registros de infracción del archivo podrían no estar ordenados.

Corrección

Para corregir infracciones de claves de índice, use Violation Detector con la opción de línea de comandos `--correct`. En el modo de corrección, Violation Detector lee el archivo de información de entrada especificado por el parámetro `correctionInputPath`. Este archivo tiene el mismo formato que el archivo `detectionOutputPath`, lo que permite utilizar el resultado de la detección como información de entrada para la corrección.

Violation Detector ofrece dos maneras distintas de corregir las infracciones de claves de índice:

- Eliminar las infracciones: se eliminan los elementos de la tabla cuyos valores de atributos infringen alguna clave.
- Actualizar las infracciones: se actualizan los elementos de la tabla, para lo cual se sustituyen los atributos infractores por otros que no causan ninguna infracción.

En ambos casos, puede utilizar el archivo de resultados del modo de detección como información de entrada para el modo de corrección.

Si continuamos con el ejemplo anterior de `ProductCatalog`, supongamos que desea eliminar el elemento infractor de la tabla. Para ello, utilizamos la línea de comandos siguiente.

```
$  java -jar ViolationDetector.jar --configFilePath config.txt --correct delete
```

En este momento, le pedirá que confirme si desea eliminar los elementos infractores.

```
Are you sure to delete all violations on the table?y/n
y
Confirmed, will delete violations on the table...
Violation correction from file started: Reading records from file: ./
gsi_violation_check.csv, will delete these records from table.
Violation correction from file finished: Violations delete: 1, Violations Update: 0
```

Ahora, tanto `ProductCatalog` como `PriceIndex` tienen el mismo número de elementos.

Trabajo con índices secundarios globales: Java

Puede utilizar la API de documentos AWS SDK para Java para crear una tabla Amazon DynamoDB con uno o varios índices secundarios globales, describir los índices de la tabla y utilizarlos para realizar consultas.

A continuación se indican los pasos comunes para las operaciones con tablas.

1. Cree una instancia de la clase `DynamoDB`.
2. Cree los objetos de solicitud correspondientes para proporcionar los parámetros obligatorios y opcionales de la operación.
3. Llame al método apropiado proporcionado por el cliente que ha creado en el paso anterior.

Temas

- Creación de una tabla con un índice secundario global
- Descripción de una tabla con un índice secundario global
- Consulta de un índice secundario local
- Ejemplo: índices secundarios globales que utilizan la API de documentos de AWS SDK para Java

Creación de una tabla con un índice secundario global

Puede crear índices secundarios globales a la vez que crea la tabla. Para ello, utilice `CreateTable` e indique las especificaciones para uno o varios índices secundarios globales. En el ejemplo de código Java siguiente, se crea una tabla para contener información sobre datos meteorológicos. La clave de partición es `Location` y la de ordenación, `Date`. Un índice secundario global denominado `PrecipIndex` permite obtener acceso rápido a los datos sobre precipitaciones de varias ubicaciones.

A continuación se indican los pasos que hay que seguir para crear una tabla con un índice secundario global mediante el API de documentos de DynamoDB.

1. Cree una instancia de la clase `DynamoDB`.
2. Cree una instancia de la clase `CreateTableRequest` para proporcionar la información de solicitud.

   Debe proporcionar el nombre de la tabla, su clave principal y los valores de rendimiento aprovisionado. Para el índice secundario global, debe proporcionar el nombre del índice, los ajustes de rendimiento aprovisionado, las definiciones de atributos de la clave de ordenación del índice, el esquema de claves del índice y la proyección de atributos.
3. Llame al método `createTable` proporcionando el objeto de solicitud como parámetro.

En el siguiente ejemplo de código Java se muestran los pasos anteriores. El código crea una tabla (`WeatherData`) con un índice secundario global (`PrecipIndex`). La clave de partición del índice es `Date` y la de ordenación, `Precipitation`. Todos los atributos de la tabla se proyectan en el índice. Los usuarios pueden consultar este índice para obtener los datos meteorológicos de una determinada fecha y, si lo desean, ordenarlos según la cantidad de precipitación.

Ya que `Precipitation` no es un atributo de clave para la tabla, no es obligatorio. Sin embargo, los elementos`WeatherData` sin `Precipitation` no aparecen en `PrecipIndex`.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

// Attribute definitions
ArrayList<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();

attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Location")
    .withAttributeType("S"));
attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Date")
    .withAttributeType("S"));
attributeDefinitions.add(new AttributeDefinition()
    .withAttributeName("Precipitation")
    .withAttributeType("N"));

// Table key schema
ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
tableKeySchema.add(new KeySchemaElement()
    .withAttributeName("Location")
    .withKeyType(KeyType.HASH));  //Partition key
tableKeySchema.add(new KeySchemaElement()
    .withAttributeName("Date")
    .withKeyType(KeyType.RANGE));  //Sort key

// PrecipIndex
GlobalSecondaryIndex precipIndex = new GlobalSecondaryIndex()
    .withIndexName("PrecipIndex")
    .withProvisionedThroughput(new ProvisionedThroughput()
        .withReadCapacityUnits((long) 10)
        .withWriteCapacityUnits((long) 1))
        .withProjection(new Projection().withProjectionType(ProjectionType.ALL));
```

```
ArrayList<KeySchemaElement> indexKeySchema = new ArrayList<KeySchemaElement>();

indexKeySchema.add(new KeySchemaElement()
    .withAttributeName("Date")
    .withKeyType(KeyType.HASH));  //Partition key
indexKeySchema.add(new KeySchemaElement()
    .withAttributeName("Precipitation")
    .withKeyType(KeyType.RANGE));  //Sort key

precipIndex.setKeySchema(indexKeySchema);

CreateTableRequest createTableRequest = new CreateTableRequest()
    .withTableName("WeatherData")
    .withProvisionedThroughput(new ProvisionedThroughput()
        .withReadCapacityUnits((long) 5)
        .withWriteCapacityUnits((long) 1))
    .withAttributeDefinitions(attributeDefinitions)
    .withKeySchema(tableKeySchema)
    .withGlobalSecondaryIndexes(precipIndex);

Table table = dynamoDB.createTable(createTableRequest);
System.out.println(table.getDescription());
```

Debe esperar hasta que DynamoDB cree la tabla y establezca el estado de esta última en `ACTIVE`. A partir de ese momento, puede comenzar a incluir elementos de datos en la tabla.

Descripción de una tabla con un índice secundario global

Para obtener información sobre los índices secundarios globales en una tabla, use `DescribeTable`. Para cada índice, puede obtener acceso a su nombre, esquema de claves y atributos proyectados.

Los siguientes son los pasos para acceder a la información global de índice secundario de una tabla.

1. Cree una instancia de la clase `DynamoDB`.
2. Cree una instancia de la clase `Table` para representar el índice que desea usar.
3. Llame al método `describe` del objeto `Table`.

En el siguiente ejemplo de código Java se muestran los pasos anteriores.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
```

```
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("WeatherData");
TableDescription tableDesc = table.describe();


Iterator<GlobalSecondaryIndexDescription> gsiIter =
 tableDesc.getGlobalSecondaryIndexes().iterator();
while (gsiIter.hasNext()) {
    GlobalSecondaryIndexDescription gsiDesc = gsiIter.next();
    System.out.println("Info for index "
         + gsiDesc.getIndexName() + ":");

    Iterator<KeySchemaElement> kseIter = gsiDesc.getKeySchema().iterator();
    while (kseIter.hasNext()) {
        KeySchemaElement kse = kseIter.next();
        System.out.printf("\t%s: %s\n", kse.getAttributeName(), kse.getKeyType());
    }
    Projection projection = gsiDesc.getProjection();
    System.out.println("\tThe projection type is: "
        + projection.getProjectionType());
    if (projection.getProjectionType().toString().equals("INCLUDE")) {
        System.out.println("\t\tThe non-key projected attributes are: "
            + projection.getNonKeyAttributes());
    }
}
```

Consulta de un índice secundario local

Puede utilizar `Query` en un índice secundario global, de un modo bastante similar al que realiza una `Query` en una tabla. Debe especificar el nombre del índice, los criterios de consulta de la clave de partición y la clave de ordenación (si procede) del índice y los atributos que desea devolver. En este ejemplo, el índice es `PrecipIndex`, cuyas clave de partición y ordenación son, respectivamente, `Date` y `Precipitation`. La consulta del índice devuelve todos los datos meteorológicos de una determinada fecha cuando el valor de Precipitation sea mayor que cero.

A continuación se indican los pasos que hay que seguir para consultar un índice secundario global mediante la API Document de AWS SDK para Java.

1. Cree una instancia de la clase `DynamoDB`.
2. Cree una instancia de la clase `Table` para representar el índice que desea usar.
3. Cree una instancia de la clase `Index` para el índice que desea consultar.

4. Llame al método `query` del objeto `Index`.

El nombre de atributo `Date` es una palabra reservada de DynamoDB. Por consiguiente, debe usar un nombre de atributo de expresión como marcador de posición en la expresión `KeyConditionExpression`.

En el siguiente ejemplo de código Java se muestran los pasos anteriores.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

Table table = dynamoDB.getTable("WeatherData");
Index index = table.getIndex("PrecipIndex");

QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("#d = :v_date and Precipitation = :v_precip")
    .withNameMap(new NameMap()
        .with("#d", "Date"))
    .withValueMap(new ValueMap()
        .withString(":v_date","2013-08-10")
        .withNumber(":v_precip",0));

ItemCollection<QueryOutcome> items = index.query(spec);
Iterator<Item> iter = items.iterator();
while (iter.hasNext()) {
    System.out.println(iter.next().toJSONPretty());
}
```

Ejemplo: índices secundarios globales que utilizan la API de documentos de AWS SDK para Java

En el siguiente ejemplo de código Java se muestra cómo usar índices secundarios globales. En el ejemplo se crea una tabla denominada `Issues`, que puede utilizarse en un sistema sencillo de seguimiento de errores con fines de desarrollo de software. La clave de partición es `IssueId` y la de ordenación, `Title`. Hay tres índices secundarios globales en esta tabla:

- `CreateDateIndex`: la clave de partición es `CreateDate` y la de ordenación `IssueId`. Además de las claves de tabla, se proyectan en el índice los atributos `Description` y `Status`.
- `TitleIndex`: la clave de partición es `Title` y la de ordenación `IssueId`. No se proyecta en el índice ningún otro atributo excepto las claves de tabla.

- `DueDateIndex`: la clave de partición es `DueDate` y no hay clave de ordenación. Todos los atributos de la tabla se proyectan en el índice.

Una vez que se ha creado la tabla `Issues`, el programa carga la tabla con datos que representan informes de errores de software. A continuación, consulta los datos utilizando los índices secundarios globales. Por último, el programa elimina la tabla `Issues`.

Para obtener instrucciones paso a paso para probar el siguiente ejemplo, consulte Ejemplos de código Java.

Example

```
package com.amazonaws.codesamples.document;

import java.util.ArrayList;
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Index;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.GlobalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;

public class DocumentAPIGlobalSecondaryIndexExample {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    public static String tableName = "Issues";
```

```
   public static void main(String[] args) throws Exception {

       createTable();
       loadData();

       queryIndex("CreateDateIndex");
       queryIndex("TitleIndex");
       queryIndex("DueDateIndex");

       deleteTable(tableName);

   }

   public static void createTable() {

       // Attribute definitions
       ArrayList<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();

       attributeDefinitions.add(new
AttributeDefinition().withAttributeName("IssueId").withAttributeType("S"));
       attributeDefinitions.add(new
AttributeDefinition().withAttributeName("Title").withAttributeType("S"));
       attributeDefinitions.add(new
AttributeDefinition().withAttributeName("CreateDate").withAttributeType("S"));
       attributeDefinitions.add(new
AttributeDefinition().withAttributeName("DueDate").withAttributeType("S"));

       // Key schema for table
       ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
       tableKeySchema.add(new
KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.HASH)); //
Partition

                    // key
       tableKeySchema.add(new
KeySchemaElement().withAttributeName("Title").withKeyType(KeyType.RANGE)); // Sort

                   // key

       // Initial provisioned throughput settings for the indexes
       ProvisionedThroughput ptIndex = new
ProvisionedThroughput().withReadCapacityUnits(1L)
```

```
                .withWriteCapacityUnits(1L);

        // CreateDateIndex
        GlobalSecondaryIndex createDateIndex = new
GlobalSecondaryIndex().withIndexName("CreateDateIndex")
                .withProvisionedThroughput(ptIndex)
                .withKeySchema(new
KeySchemaElement().withAttributeName("CreateDate").withKeyType(KeyType.HASH), //
Partition

                          // key
                        new
KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.RANGE)) // Sort

                 // key
                .withProjection(
                        new
Projection().withProjectionType("INCLUDE").withNonKeyAttributes("Description",
"Status"));

        // TitleIndex
        GlobalSecondaryIndex titleIndex = new
GlobalSecondaryIndex().withIndexName("TitleIndex")
                .withProvisionedThroughput(ptIndex)
                .withKeySchema(new
KeySchemaElement().withAttributeName("Title").withKeyType(KeyType.HASH), // Partition

                     // key
                        new
KeySchemaElement().withAttributeName("IssueId").withKeyType(KeyType.RANGE)) // Sort

                 // key
                .withProjection(new Projection().withProjectionType("KEYS_ONLY"));

        // DueDateIndex
        GlobalSecondaryIndex dueDateIndex = new
GlobalSecondaryIndex().withIndexName("DueDateIndex")
                .withProvisionedThroughput(ptIndex)
                .withKeySchema(new
KeySchemaElement().withAttributeName("DueDate").withKeyType(KeyType.HASH)) //
Partition

                       // key
                .withProjection(new Projection().withProjectionType("ALL"));
```

```
        CreateTableRequest createTableRequest = new
CreateTableRequest().withTableName(tableName)
                .withProvisionedThroughput(
                        new ProvisionedThroughput().withReadCapacityUnits((long)
1).withWriteCapacityUnits((long) 1))

.withAttributeDefinitions(attributeDefinitions).withKeySchema(tableKeySchema)
                .withGlobalSecondaryIndexes(createDateIndex, titleIndex, dueDateIndex);

        System.out.println("Creating table " + tableName + "...");
        dynamoDB.createTable(createTableRequest);

        // Wait for table to become active
        System.out.println("Waiting for " + tableName + " to become ACTIVE...");
        try {
            Table table = dynamoDB.getTable(tableName);
            table.waitForActive();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public static void queryIndex(String indexName) {

        Table table = dynamoDB.getTable(tableName);


System.out.println("\n***********************************************************\n");
        System.out.print("Querying index " + indexName + "...");

        Index index = table.getIndex(indexName);

        ItemCollection<QueryOutcome> items = null;

        QuerySpec querySpec = new QuerySpec();

        if (indexName == "CreateDateIndex") {
            System.out.println("Issues filed on 2013-11-01");
            querySpec.withKeyConditionExpression("CreateDate = :v_date and
begins_with(IssueId, :v_issue)")
                    .withValueMap(new ValueMap().withString(":v_date",
"2013-11-01").withString(":v_issue", "A-"));
            items = index.query(querySpec);
```

```
        } else if (indexName == "TitleIndex") {
            System.out.println("Compilation errors");
            querySpec.withKeyConditionExpression("Title = :v_title and
begins_with(IssueId, :v_issue)")
                    .withValueMap(
                            new ValueMap().withString(":v_title", "Compilation
error").withString(":v_issue", "A-"));
            items = index.query(querySpec);
        } else if (indexName == "DueDateIndex") {
            System.out.println("Items that are due on 2013-11-30");
            querySpec.withKeyConditionExpression("DueDate = :v_date")
                    .withValueMap(new ValueMap().withString(":v_date", "2013-11-30"));
            items = index.query(querySpec);
        } else {
            System.out.println("\nNo valid index name provided");
            return;
        }

        Iterator<Item> iterator = items.iterator();

        System.out.println("Query: printing results...");

        while (iterator.hasNext()) {
            System.out.println(iterator.next().toJSONPretty());
        }

    }

    public static void deleteTable(String tableName) {

        System.out.println("Deleting table " + tableName + "...");

        Table table = dynamoDB.getTable(tableName);
        table.delete();

        // Wait for table to be deleted
        System.out.println("Waiting for " + tableName + " to be deleted...");
        try {
            table.waitForDelete();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

```
    public static void loadData() {

        System.out.println("Loading data into table " + tableName + "...");

        // IssueId, Title,
        // Description,
        // CreateDate, LastUpdateDate, DueDate,
        // Priority, Status

        putItem("A-101", "Compilation error", "Can't compile Project X - bad version
number. What does this mean?",
                "2013-11-01", "2013-11-02", "2013-11-10", 1, "Assigned");

        putItem("A-102", "Can't read data file", "The main data file is missing, or the
permissions are incorrect",
                "2013-11-01", "2013-11-04", "2013-11-30", 2, "In progress");

        putItem("A-103", "Test failure", "Functional test of Project X produces
errors", "2013-11-01", "2013-11-02",
                "2013-11-10", 1, "In progress");

        putItem("A-104", "Compilation error", "Variable 'messageCount' was not
initialized.", "2013-11-15",
                "2013-11-16", "2013-11-30", 3, "Assigned");

        putItem("A-105", "Network issue", "Can't ping IP address 127.0.0.1. Please fix
this.", "2013-11-15",
                "2013-11-16", "2013-11-19", 5, "Assigned");

    }

    public static void putItem(

            String issueId, String title, String description, String createDate, String
lastUpdateDate, String dueDate,
            Integer priority, String status) {

        Table table = dynamoDB.getTable(tableName);

        Item item = new Item().withPrimaryKey("IssueId", issueId).withString("Title",
title)
                .withString("Description", description).withString("CreateDate",
createDate)
```

```
                .withString("LastUpdateDate", lastUpdateDate).withString("DueDate",
 dueDate)
                .withNumber("Priority", priority).withString("Status", status);

        table.putItem(item);
    }

}
```

Trabajar con índices secundarios globales: .NET

Puede utilizar la API de bajo nivel AWS SDK para .NET para crear una tabla Amazon DynamoDB con uno o varios índices secundarios globales, describir los índices de la tabla y utilizarlos para realizar consultas. Estas operaciones se mapean a las operaciones correspondientes de DynamoDB. Para obtener más información, consulte la Referencia de la API de Amazon DynamoDB.

A continuación se indican los pasos comunes para las operaciones con tablas mediante el API de bajo nivel de .NET.

1. Cree una instancia de la clase `AmazonDynamoDBClient`.
2. Cree los objetos de solicitud correspondientes para proporcionar los parámetros obligatorios y opcionales de la operación.

   Por ejemplo, cree un objeto `CreateTableRequest` para crear una tabla y un objeto `QueryRequest` para consultar una tabla o un índice.
3. Ejecute el método apropiado proporcionado por el cliente que ha creado en el paso anterior.

Temas

- Creación de una tabla con un índice secundario global
- Descripción de una tabla con un índice secundario global
- Consulta de un índice secundario local
- Ejemplo: índices secundarios globales que usan la API de bajo nivel de AWS SDK para .NET

Creación de una tabla con un índice secundario global

Puede crear índices secundarios globales a la vez que crea la tabla. Para ello, utilice `CreateTable` e indique las especificaciones para uno o varios índices secundarios globales. En el ejemplo de

código C# siguiente, se crea una tabla para contener información sobre datos meteorológicos. La clave de partición es `Location` y la de ordenación, `Date`. Un índice secundario global denominado `PrecipIndex` permite obtener acceso rápido a los datos sobre precipitaciones de varias ubicaciones.

A continuación se indican los pasos que hay que seguir para crear una tabla con un índice secundario global mediante el API de bajo nivel de .NET.

1. Cree una instancia de la clase `AmazonDynamoDBClient`.
2. Cree una instancia de la clase `CreateTableRequest` para proporcionar la información de solicitud.

   Debe proporcionar el nombre de la tabla, su clave principal y los valores de rendimiento aprovisionado. Para el índice secundario global, debe proporcionar el nombre del índice, los ajustes de rendimiento aprovisionado, las definiciones de atributos de la clave de ordenación del índice, el esquema de claves del índice y la proyección de atributos.
3. Ejecute el método `CreateTable` proporcionando el objeto de solicitud como parámetro.

En el siguiente ejemplo de código C# se ponen en práctica los pasos anteriores. El código crea una tabla (`WeatherData`) con un índice secundario global (`PrecipIndex`). La clave de partición del índice es `Date` y la de ordenación, `Precipitation`. Todos los atributos de la tabla se proyectan en el índice. Los usuarios pueden consultar este índice para obtener los datos meteorológicos de una determinada fecha y, si lo desean, ordenarlos según la cantidad de precipitación.

Ya que `Precipitation` no es un atributo de clave para la tabla, no es obligatorio. Sin embargo, los elementos`WeatherData` sin `Precipitation` no aparecen en `PrecipIndex`.

```
client = new AmazonDynamoDBClient();
string tableName = "WeatherData";

// Attribute definitions
var attributeDefinitions = new List<AttributeDefinition>()
{
    {new AttributeDefinition{
        AttributeName = "Location",
        AttributeType = "S"}},
    {new AttributeDefinition{
        AttributeName = "Date",
        AttributeType = "S"}},
    {new AttributeDefinition(){
```

```
        AttributeName = "Precipitation",
        AttributeType = "N"}
    }
};

// Table key schema
var tableKeySchema = new List<KeySchemaElement>()
{
    {new KeySchemaElement {
        AttributeName = "Location",
        KeyType = "HASH"}},  //Partition key
    {new KeySchemaElement {
        AttributeName = "Date",
        KeyType = "RANGE"}  //Sort key
    }
};

// PrecipIndex
var precipIndex = new GlobalSecondaryIndex
{
    IndexName = "PrecipIndex",
    ProvisionedThroughput = new ProvisionedThroughput
    {
        ReadCapacityUnits = (long)10,
        WriteCapacityUnits = (long)1
    },
    Projection = new Projection { ProjectionType = "ALL" }
};

var indexKeySchema = new List<KeySchemaElement> {
    {new KeySchemaElement { AttributeName = "Date", KeyType = "HASH"}},  //Partition
 key
    {new KeySchemaElement{AttributeName = "Precipitation",KeyType = "RANGE"}}  //Sort
 key
};

precipIndex.KeySchema = indexKeySchema;

CreateTableRequest createTableRequest = new CreateTableRequest
{
    TableName = tableName,
    ProvisionedThroughput = new ProvisionedThroughput
    {
        ReadCapacityUnits = (long)5,
```

```
        WriteCapacityUnits = (long)1
    },
    AttributeDefinitions = attributeDefinitions,
    KeySchema = tableKeySchema,
    GlobalSecondaryIndexes = { precipIndex }
};

CreateTableResponse response = client.CreateTable(createTableRequest);
Console.WriteLine(response.CreateTableResult.TableDescription.TableName);
Console.WriteLine(response.CreateTableResult.TableDescription.TableStatus);
```

Debe esperar hasta que DynamoDB cree la tabla y establezca el estado de esta última en `ACTIVE`. A partir de ese momento, puede comenzar a incluir elementos de datos en la tabla.

Descripción de una tabla con un índice secundario global

Para obtener información sobre los índices secundarios globales en una tabla, use `DescribeTable`. Para cada índice, puede obtener acceso a su nombre, esquema de claves y atributos proyectados.

A continuación se indican los pasos que hay que seguir para acceder a la información de un índice secundario global de una tabla mediante el API de bajo nivel de .NET.

1. Cree una instancia de la clase `AmazonDynamoDBClient`.
2. Ejecute el método `describeTable` proporcionando el objeto de solicitud como parámetro.

   Cree una instancia de la clase `DescribeTableRequest` para proporcionar la información de solicitud. Debe proporcionar el nombre de la tabla.
3.

En el siguiente ejemplo de código C# se ponen en práctica los pasos anteriores.

Example

```
client = new AmazonDynamoDBClient();
string tableName = "WeatherData";

DescribeTableResponse response = client.DescribeTable(new DescribeTableRequest
 { TableName = tableName});

List<GlobalSecondaryIndexDescription> globalSecondaryIndexes =
```

```
response.DescribeTableResult.Table.GlobalSecondaryIndexes;

// This code snippet will work for multiple indexes, even though
// there is only one index in this example.

foreach (GlobalSecondaryIndexDescription gsiDescription in globalSecondaryIndexes) {
     Console.WriteLine("Info for index " + gsiDescription.IndexName + ":");

     foreach (KeySchemaElement kse in gsiDescription.KeySchema) {
          Console.WriteLine("\t" + kse.AttributeName + ": key type is " + kse.KeyType);
     }

      Projection projection = gsiDescription.Projection;
      Console.WriteLine("\tThe projection type is: " + projection.ProjectionType);

      if (projection.ProjectionType.ToString().Equals("INCLUDE")) {
           Console.WriteLine("\t\tThe non-key projected attributes are: "
                + projection.NonKeyAttributes);
      }
}
```

Consulta de un índice secundario local

Puede utilizar `Query` en un índice secundario global, de un modo bastante similar al que realiza una `Query` en una tabla. Debe especificar el nombre del índice, los criterios de consulta de la clave de partición y la clave de ordenación (si procede) del índice y los atributos que desea devolver. En este ejemplo, el índice es `PrecipIndex`, cuyas clave de partición y ordenación son, respectivamente, `Date` y `Precipitation`. La consulta del índice devuelve todos los datos meteorológicos de una determinada fecha cuando el valor de Precipitation sea mayor que cero.

A continuación se indican los pasos que hay que seguir para consultar un índice secundario global mediante el API de bajo nivel de .NET.

1. Cree una instancia de la clase `AmazonDynamoDBClient`.
2. Cree una instancia de la clase `QueryRequest` para proporcionar la información de solicitud.
3. Ejecute el método `query` proporcionando el objeto de solicitud como parámetro.

El nombre de atributo `Date` es una palabra reservada de DynamoDB. Por consiguiente, debe usar un nombre de atributo de expresión como marcador de posición en la expresión `KeyConditionExpression`.

En el siguiente ejemplo de código C# se ponen en práctica los pasos anteriores.

Example

```
client = new AmazonDynamoDBClient();

QueryRequest queryRequest = new QueryRequest
{
    TableName = "WeatherData",
    IndexName = "PrecipIndex",
    KeyConditionExpression = "#dt = :v_date and Precipitation > :v_precip",
    ExpressionAttributeNames = new Dictionary<String, String> {
        {"#dt", "Date"}
    },
    ExpressionAttributeValues = new Dictionary<string, AttributeValue> {
        {":v_date", new AttributeValue { S =  "2013-08-01" }},
        {":v_precip", new AttributeValue { N =  "0" }}
    },
    ScanIndexForward = true
};

var result = client.Query(queryRequest);

var items = result.Items;
foreach (var currentItem in items)
{
    foreach (string attr in currentItem.Keys)
    {
        Console.Write(attr + "---> ");
        if (attr == "Precipitation")
        {
            Console.WriteLine(currentItem[attr].N);
    }
    else
    {
        Console.WriteLine(currentItem[attr].S);
    }

         }
     Console.WriteLine();
}
```

Ejemplo: índices secundarios globales que usan la API de bajo nivel de AWS SDK para .NET

En el siguiente ejemplo de código C# se muestra cómo usar índices secundarios globales. En el ejemplo se crea una tabla denominada `Issues`, que puede utilizarse en un sistema sencillo de seguimiento de errores con fines de desarrollo de software. La clave de partición es `IssueId` y la de ordenación, `Title`. Hay tres índices secundarios globales en esta tabla:

- `CreateDateIndex`: la clave de partición es `CreateDate` y la de ordenación `IssueId`. Además de las claves de tabla, se proyectan en el índice los atributos `Description` y `Status`.
- `TitleIndex`: la clave de partición es `Title` y la de ordenación `IssueId`. No se proyecta en el índice ningún otro atributo excepto las claves de tabla.
- `DueDateIndex`: la clave de partición es `DueDate` y no hay clave de ordenación. Todos los atributos de la tabla se proyectan en el índice.

Una vez que se ha creado la tabla `Issues`, el programa carga la tabla con datos que representan informes de errores de software. A continuación, consulta los datos utilizando los índices secundarios globales. Por último, el programa elimina la tabla `Issues`.

Para obtener instrucciones paso a paso para probar el siguiente ejemplo, consulte Ejemplos de código .NET.

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelGlobalSecondaryIndexExample
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        public static String tableName = "Issues";
```

```
        public static void Main(string[] args)
        {
            CreateTable();
            LoadData();

            QueryIndex("CreateDateIndex");
            QueryIndex("TitleIndex");
            QueryIndex("DueDateIndex");

            DeleteTable(tableName);

            Console.WriteLine("To continue, press enter");
            Console.Read();
        }

        private static void CreateTable()
        {
            // Attribute definitions
            var attributeDefinitions = new List<AttributeDefinition>()
        {
            {new AttributeDefinition {
                 AttributeName = "IssueId", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "Title", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "CreateDate", AttributeType = "S"
             }},
            {new AttributeDefinition {
                 AttributeName = "DueDate", AttributeType = "S"
             }}
        };

            // Key schema for table
            var tableKeySchema = new List<KeySchemaElement>() {
            {
                new KeySchemaElement {
                    AttributeName= "IssueId",
                    KeyType = "HASH" //Partition key
                }
            },
            {
                new KeySchemaElement {
```

```
                AttributeName = "Title",
                KeyType = "RANGE" //Sort key
            }
        }
    };

        // Initial provisioned throughput settings for the indexes
        var ptIndex = new ProvisionedThroughput
        {
            ReadCapacityUnits = 1L,
            WriteCapacityUnits = 1L
        };

        // CreateDateIndex
        var createDateIndex = new GlobalSecondaryIndex()
        {
            IndexName = "CreateDateIndex",
            ProvisionedThroughput = ptIndex,
            KeySchema = {
            new KeySchemaElement {
                AttributeName = "CreateDate", KeyType = "HASH" //Partition key
            },
            new KeySchemaElement {
                AttributeName = "IssueId", KeyType = "RANGE" //Sort key
            }
        },
            Projection = new Projection
            {
                ProjectionType = "INCLUDE",
                NonKeyAttributes = {
                "Description", "Status"
            }
            }
        };

        // TitleIndex
        var titleIndex = new GlobalSecondaryIndex()
        {
            IndexName = "TitleIndex",
            ProvisionedThroughput = ptIndex,
            KeySchema = {
            new KeySchemaElement {
                AttributeName = "Title", KeyType = "HASH" //Partition key
            },
```

```
            new KeySchemaElement {
                AttributeName = "IssueId", KeyType = "RANGE" //Sort key
            }
        },
            Projection = new Projection
            {
                ProjectionType = "KEYS_ONLY"
            }
        };

        // DueDateIndex
        var dueDateIndex = new GlobalSecondaryIndex()
        {
            IndexName = "DueDateIndex",
            ProvisionedThroughput = ptIndex,
            KeySchema = {
            new KeySchemaElement {
                AttributeName = "DueDate",
                KeyType = "HASH" //Partition key
            }
        },
            Projection = new Projection
            {
                ProjectionType = "ALL"
            }
        };



        var createTableRequest = new CreateTableRequest
        {
            TableName = tableName,
            ProvisionedThroughput = new ProvisionedThroughput
            {
                ReadCapacityUnits = (long)1,
                WriteCapacityUnits = (long)1
            },
            AttributeDefinitions = attributeDefinitions,
            KeySchema = tableKeySchema,
            GlobalSecondaryIndexes = {
            createDateIndex, titleIndex, dueDateIndex
        }
        };
```

```
            Console.WriteLine("Creating table " + tableName + "...");
            client.CreateTable(createTableRequest);

            WaitUntilTableReady(tableName);
        }

        private static void LoadData()
        {
            Console.WriteLine("Loading data into table " + tableName + "...");

            // IssueId, Title,
            // Description,
            // CreateDate, LastUpdateDate, DueDate,
            // Priority, Status

            putItem("A-101", "Compilation error",
                "Can't compile Project X - bad version number. What does this mean?",
                "2013-11-01", "2013-11-02", "2013-11-10",
                1, "Assigned");

            putItem("A-102", "Can't read data file",
                "The main data file is missing, or the permissions are incorrect",
                "2013-11-01", "2013-11-04", "2013-11-30",
                2, "In progress");

            putItem("A-103", "Test failure",
                "Functional test of Project X produces errors",
                "2013-11-01", "2013-11-02", "2013-11-10",
                1, "In progress");

            putItem("A-104", "Compilation error",
                "Variable 'messageCount' was not initialized.",
                "2013-11-15", "2013-11-16", "2013-11-30",
                3, "Assigned");

            putItem("A-105", "Network issue",
                "Can't ping IP address 127.0.0.1. Please fix this.",
                "2013-11-15", "2013-11-16", "2013-11-19",
                5, "Assigned");
        }

        private static void putItem(
            String issueId, String title,
            String description,
```

```
            String createDate, String lastUpdateDate, String dueDate,
            Int32 priority, String status)
        {
            Dictionary<String, AttributeValue> item = new Dictionary<string,
AttributeValue>();

            item.Add("IssueId", new AttributeValue
            {
                S = issueId
            });
            item.Add("Title", new AttributeValue
            {
                S = title
            });
            item.Add("Description", new AttributeValue
            {
                S = description
            });
            item.Add("CreateDate", new AttributeValue
            {
                S = createDate
            });
            item.Add("LastUpdateDate", new AttributeValue
            {
                S = lastUpdateDate
            });
            item.Add("DueDate", new AttributeValue
            {
                S = dueDate
            });
            item.Add("Priority", new AttributeValue
            {
                N = priority.ToString()
            });
            item.Add("Status", new AttributeValue
            {
                S = status
            });

            try
            {
                client.PutItem(new PutItemRequest
                {
                    TableName = tableName,
```

```
                    Item = item
                });
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
        }

        private static void QueryIndex(string indexName)
        {
            Console.WriteLine
                ("\n*****************************************************\n");
            Console.WriteLine("Querying index " + indexName + "...");

            QueryRequest queryRequest = new QueryRequest
            {
                TableName = tableName,
                IndexName = indexName,
                ScanIndexForward = true
            };


            String keyConditionExpression;
            Dictionary<string, AttributeValue> expressionAttributeValues = new
Dictionary<string, AttributeValue>();

            if (indexName == "CreateDateIndex")
            {
                Console.WriteLine("Issues filed on 2013-11-01\n");

                keyConditionExpression = "CreateDate = :v_date and
begins_with(IssueId, :v_issue)";
                expressionAttributeValues.Add(":v_date", new AttributeValue
                {
                    S = "2013-11-01"
                });
                expressionAttributeValues.Add(":v_issue", new AttributeValue
                {
                    S = "A-"
                });
            }
            else if (indexName == "TitleIndex")
            {
```

```
                Console.WriteLine("Compilation errors\n");

                keyConditionExpression = "Title = :v_title and
begins_with(IssueId, :v_issue)";
                expressionAttributeValues.Add(":v_title", new AttributeValue
                {
                    S = "Compilation error"
                });
                expressionAttributeValues.Add(":v_issue", new AttributeValue
                {
                    S = "A-"
                });

                // Select
                queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
            }
            else if (indexName == "DueDateIndex")
            {
                Console.WriteLine("Items that are due on 2013-11-30\n");

                keyConditionExpression = "DueDate = :v_date";
                expressionAttributeValues.Add(":v_date", new AttributeValue
                {
                    S = "2013-11-30"
                });

                // Select
                queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
            }
            else
            {
                Console.WriteLine("\nNo valid index name provided");
                return;
            }

            queryRequest.KeyConditionExpression = keyConditionExpression;
            queryRequest.ExpressionAttributeValues = expressionAttributeValues;

            var result = client.Query(queryRequest);
            var items = result.Items;
            foreach (var currentItem in items)
            {
                foreach (string attr in currentItem.Keys)
                {
```

```
                if (attr == "Priority")
                {
                    Console.WriteLine(attr + "---> " + currentItem[attr].N);
                }
                else
                {
                    Console.WriteLine(attr + "---> " + currentItem[attr].S);
                }
            }
            Console.WriteLine();
        }
    }

    private static void DeleteTable(string tableName)
    {
        Console.WriteLine("Deleting table " + tableName + "...");
        client.DeleteTable(new DeleteTableRequest
        {
            TableName = tableName
        });
        WaitForTableToBeDeleted(tableName);
    }

    private static void WaitUntilTableReady(string tableName)
    {
        string status = null;
        // Let us wait until table is created. Call DescribeTable.
        do
        {
            System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
            try
            {
                var res = client.DescribeTable(new DescribeTableRequest
                {
                    TableName = tableName
                });

                Console.WriteLine("Table name: {0}, status: {1}",
                          res.Table.TableName,
                          res.Table.TableStatus);
                status = res.Table.TableStatus;
            }
            catch (ResourceNotFoundException)
            {
```

```
                    // DescribeTable is eventually consistent. So you might
                    // get resource not found. So we handle the potential exception.
                }
            } while (status != "ACTIVE");
        }

        private static void WaitForTableToBeDeleted(string tableName)
        {
            bool tablePresent = true;

            while (tablePresent)
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                }
                catch (ResourceNotFoundException)
                {
                    tablePresent = false;
                }
            }
        }
    }
}
```

Uso de índices secundarios globales en DynamoDB con la AWS CLI

Puede utilizar la AWS CLI para crear una tabla de Amazon DynamoDB con uno o varios índices secundarios globales, describir los índices de la tabla y utilizarlos para realizar consultas.

Temas

- Creación de una tabla con un índice secundario global
- Adición de un índice secundario global a una tabla existente

- Descripción de una tabla con un índice secundario global
- Consulta de un índice secundario local

Creación de una tabla con un índice secundario global

Se pueden crear índices secundarios globales al mismo tiempo que se crea una tabla. Para ello, utilice el parámetro `create-table` y proporcione sus especificaciones para uno o más índices secundarios globales. En el siguiente ejemplo, se crea una tabla llamada `GameScores` con un índice secundario global `GameTitleIndex`. La tabla base tiene una clave de partición de `UserId` y una clave de ordenación de `GameTitle`, lo que le permite encontrar eficientemente la mejor puntuación de un usuario individual para un juego específico, mientras que el GSI tiene una clave de partición de `GameTitle` y una clave de ordenación de `TopScore`, lo que te permite encontrar rápidamente la puntuación más alta en general para un juego en particular.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=UserId,AttributeType=S \
                            AttributeName=GameTitle,AttributeType=S \
                            AttributeName=TopScore,AttributeType=N  \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                                {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                \"Projection\":{
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
        ]"
```

Debe esperar hasta que DynamoDB cree la tabla y establezca el estado de esta última en `ACTIVE`. A partir de ese momento, puede comenzar a incluir elementos de datos en la tabla. Puede utilizar describe-table Para determinar el estado de la creación de tablas.

Adición de un índice secundario global a una tabla existente

Los índices secundarios globales también se pueden agregar o modificar después de la creación de la tabla. Para ello, utilice el parámetro `update-table` y proporcione sus especificaciones para uno o más índices secundarios globales. En el siguiente ejemplo se utiliza el mismo esquema que el ejemplo anterior, pero se supone que la tabla ya se ha creado y que vamos a agregar el GSI más adelante.

```
aws dynamodb update-table \
    --table-name GameScores \
    --attribute-definitions AttributeName=TopScore,AttributeType=N  \
    --global-secondary-index-updates \
        "[
            {
                \"Create\": {
                    \"IndexName\": \"GameTitleIndex\",
                    \"KeySchema\": [{\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH
\"},
                                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE
\"}],
                    \"Projection\":{
                        \"ProjectionType\":\"INCLUDE\",
                        \"NonKeyAttributes\":[\"UserId\"]
                    }
                }
            }
        ]"
```

Descripción de una tabla con un índice secundario global

Para obtener información acerca de los índices secundarios globales en una tabla, utilice el parámetro `describe-table`. Para cada índice, puede obtener acceso a su nombre, esquema de claves y atributos proyectados.

```
aws dynamodb describe-table --table-name GameScores
```

Consulta de un índice secundario local

Puede utilizar la operación `query` en un índice secundario global de un modo bastante similar a como se usa `query` en una tabla. Debe especificar el nombre del índice, los criterios de consulta de la clave de ordenación del índice y los atributos que desea devolver. En este ejemplo, el índice es `GameTitleIndex` y la clave de ordenación del índice es `GameTitle`.

Los únicos atributos devueltos son aquellos que se han proyectado en el índice. Puede modificar esta consulta de modo que también seleccione atributos sin clave, pero esto requeriría realizar actividad de recuperación en la tabla, lo que resulta relativamente costoso. Para obtener más información sobre recuperaciones de tablas, consulte Proyecciones de atributos.

```
aws dynamodb query --table-name GameScores\
    --index-name GameTitleIndex \
    --key-condition-expression "GameTitle = :v_game" \
    --expression-attribute-values '{":v_game":{"S":"Alien Adventure"} }'
```

Índices secundarios locales

Algunas aplicaciones solo necesitan consultar datos mediante la clave principal de la tabla base. Sin embargo, puede haber situaciones en las que una clave de ordenación alternativa sería útil. Para que la aplicación disponga de diversas claves de ordenación entre las que elegir, puede crear uno o varios índices secundarios en una tabla de Amazon DynamoDB y emitir solicitudes `Query` o `Scan` para estos índices.

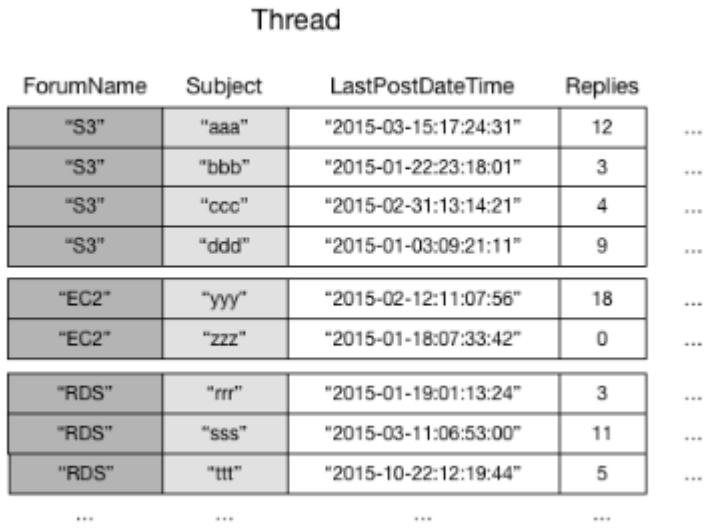
Temas

- Situación: Uso de un índice secundario local
- Proyecciones de atributos
- Creación de un índice secundario local
- Lectura de datos de un índice secundario local
- Escrituras de elementos e índices secundarios locales
- Consideraciones sobre el rendimiento aprovisionado para los índices secundarios locales
- Consideraciones sobre el almacenamiento para los índices secundarios locales
- Colecciones de elementos en los índices secundarios locales
- Trabajar con índices secundarios locales: Java
- Trabajar con índices secundarios locales: .NET

- Uso de índices secundarios locales en la AWS CLI de DynamoDB

Situación: Uso de un índice secundario local

Como ejemplo, veamos la tabla `Thread`. Esta tabla es útil para una aplicación como, por ejemplo, Foros de debate de AWS. En el siguiente diagrama se muestra cómo se organizarían los elementos de la tabla. No se muestran todos los atributos.



DynamoDB almacena todos los elementos que tienen el mismo valor de clave de partición de forma continua. En este ejemplo, para un valor determinado de `ForumName`, una operación `Query` podría localizar inmediatamente todas las conversaciones de ese foro. Dentro de un grupo de elementos con el mismo valor de clave de partición, los elementos se ordenan según el valor de la clave de ordenación. Si la clave de ordenación (`Subject`) también se incluye en la consulta, DynamoDB puede afinar los resultados que se devuelven; por ejemplo, devolvería todas las conversaciones del foro "S3" cuyo valor de `Subject` comience por la letra "a".

Algunas solicitudes podrían requerir patrones más complejos de acceso a los datos. Por ejemplo:

- ¿Qué conversaciones del foro son las que suscitan más visualizaciones y respuestas?
- ¿Qué conversación de un foro determinado tiene mayor cantidad de mensajes?
- ¿Cuántas conversaciones se han publicado en un foro determinado dentro de un periodo concreto?

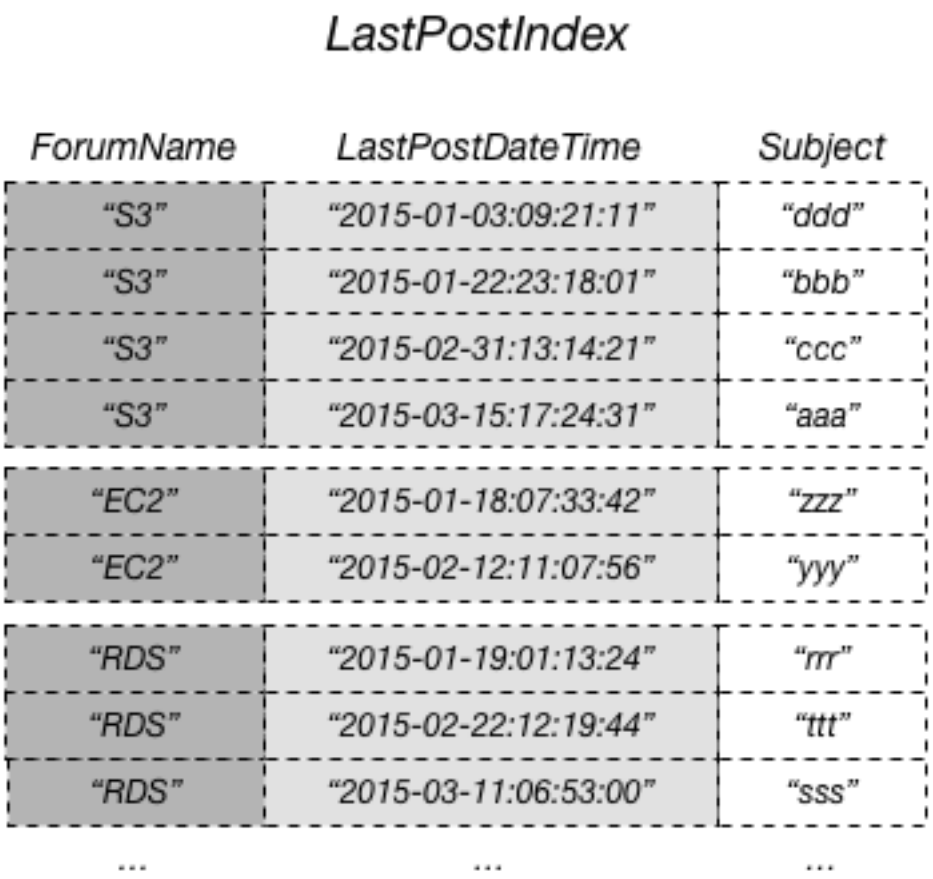
Para responder a estas preguntas, la acción `Query` no bastaría. En su lugar, habría que aplicar `Scan` a la tabla completa. Si la tabla contiene millones de elementos, esto consumiría una cantidad enorme de desempeño de lectura provisionado y tardaría mucho tiempo en completarse.

Sin embargo, puede especificar uno o más índices secundarios locales en atributos que no sean clave, tales como `Replies` or `LastPostDateTime`.

Un índice secundario local mantiene una clave de ordenación alternativa para un determinado valor de clave de partición. Un índice secundario local también contiene una copia parcial o total de los atributos de la tabla base. Al crear la tabla, se especifican qué atributos se proyectarán en el índice secundario local. Los datos de un índice secundario local se organizan según la misma clave de partición que la tabla base, pero con una clave de ordenación distinta. Esto le permite obtener acceso a los elementos de datos de forma eficiente en esta otra dimensión. Para lograr una mayor flexibilidad de consulta o examen, puede crear hasta cinco índices secundarios locales por tabla.

Supongamos que una aplicación tiene que encontrar todas las conversaciones que se han publicado en los últimos tres meses en un foro particular. Sin un índice secundario local, la aplicación tendría que aplicar la operación `Scan` a toda la tabla `Thread` y, después, descartar aquellas publicaciones que no estuviesen comprendidos en el periodo especificado. Con un índice secundario local una operación `Query` podría usar `LastPostDateTime` como clave de ordenación para encontrar rápidamente los datos.

En el siguiente diagrama se muestra un índice secundario local denominado `LastPostIndex`. Tenga en cuenta que la clave de partición es la misma que para la tabla `Thread`, pero que la clave de ordenación es `LastPostDateTime`.



Cada índice secundario local debe cumplir las condiciones siguientes:

- La clave de partición ha de ser la misma que aquella de la tabla base.
- La clave de ordenación consta exactamente de un atributo escalar.
- La clave de ordenación de la tabla base se proyectará en el índice, donde actuará como atributo sin clave.

En este ejemplo, la clave de partición es `ForumName` y la clave de ordenación del índice secundario local es `LastPostDateTime`. Además, el valor de clave de ordenación de la tabla base (en este ejemplo, `Subject`) se proyecta en el índice, aunque no forma parte de la clave de índice. Si una aplicación requiere una lista basada en `ForumName` y `LastPostDateTime`, puede emitir una solicitud `Query` para `LastPostIndex`. Los resultados de la consulta se ordenan por `LastPostDateTime` y se pueden devolver en orden ascendente o descendente. La consulta también puede aplicar condiciones de clave, tales como devolver solamente aquellos elementos cuyo valor de `LastPostDateTime` esté comprendido en un periodo determinado.

Cada índice secundario local contiene automáticamente las claves de partición y ordenación de su tabla base. Si lo desea, puede proyectar atributos sin clave en el índice. Cuando se consulta el índice, DynamoDB puede recuperar estos atributos proyectados de forma eficiente. Cuando se consulta un índice secuandario local, la consulta también puede recuperar atributos no proyectados en el índice. DynamoDB recuperará automáticamente estos atributos de la tabla base, pero con una mayor latencia y costes más elevados de rendimiento aprovisionado.

Para cualquier índice secundario local, puede almacenar hasta 10 GB de datos por cada valor diferente de clave de partición. Esta cifra incluye todos los elementos de la tabla base, además de todos los elementos de los índices, que tengan el mismo valor de clave de partición. Para obtener más información, consulte Colecciones de elementos en los índices secundarios locales.

Proyecciones de atributos

Con `LastPostIndex`, una aplicación podría usar `ForumName` y `LastPostDateTime` como criterios de consulta. Sin embargo, para recuperar cualquier atributo adicional, DynamoDB debe realizar operaciones de lectura adicionales en la tabla `Thread`. Estas lecturas adicionales se denominan recuperaciones y pueden aumentar la cantidad total de desempeño provisionado necesario para una consulta.

Supongamos que desea rellenar una página web con una lista de todas las conversaciones del foro "S3" y el número de respuestas de cada conversación, ordenadas según la fecha y hora de la última respuesta y comenzando por la respuesta más reciente. Para rellenar esta lista, se requieren los siguientes atributos:

- `Subject`
- `Replies`
- `LastPostDateTime`

La forma más eficiente de consultar estos datos y evitar operaciones de recuperación, sería proyectar el atributo `Replies`de la tabla en el índice secundario global, tal y como se muestra en este diagrama.

LastPostIndex

ForumName	LastPostDateTime	Subject	Replies
"S3"	"2015-01-03:09:21:11"	"ddd"	9
"S3"	"2015-01-22:23:18:01"	"bbb"	3
"S3"	"2015-02-31:13:14:21"	"ccc"	4
"S3"	"2015-03-15:17:24:31"	"aaa"	12
"EC2"	"2015-01-18:07:33:42"	"zzz"	0
"EC2"	"2015-02-12:11:07:56"	"yyy"	18
"RDS"	"2015-01-19:01:13:24"	"rrr"	3
"RDS"	"2015-02-22:12:19:44"	"ttt"	5
"RDS"	"2015-03-11:06:53:00"	"sss"	11
...	...	...	...

Una proyección es el conjunto de atributos que se copia de una tabla en un índice secundario. La clave de partición y la clave de ordenación de la tabla siempre se proyectan en el índice; puede proyectar otros atributos para admitir los requisitos de consulta de la aplicación. Cuando consulta un índice, Amazon DynamoDB puede acceder a cualquier atributo de la proyección como si esos atributos estuvieran en una tabla propia.

Al crear un índice secundario, debe especificar los atributos que se proyectarán en el índice. DynamoDB ofrece tres opciones diferentes para esto:

- KEYS_ONLY: cada elemento del índice consta únicamente de los valores de la clave de partición y la clave de ordenación de la tabla, así como de los valores de las claves del índice. La opción `KEYS_ONLY` da como resultado el índice secundario más pequeño posible.
- INCLUDE: además de los atributos que se describen en `KEYS_ONLY`, el índice secundario incluirá otros atributos sin clave que se especifiquen.

- ALL: el índice secundario incluye todos los atributos de la tabla de origen. Debido a que todos los datos de la tabla están duplicados en el índice, un resultado de proyección `ALL` en el índice secundario más grande posible.

En el diagrama anterior, el atributo sin clave `Replies` se proyecta en `LastPostIndex`. Una aplicación puede consultar `LastPostIndex` en lugar de `Thread` completo para rellenar una página web con `Subject`, `Replies` y `LastPostDateTime`. Si se solicitasen otros atributos sin clave, DynamoDB tendría que recuperarlos en la tabla `Thread`.

Desde el punto de vista de una aplicación, la recuperación de atributos adicionales de la tabla base se lleva a cabo de forma automática y transparente, por lo que no es necesario volver a escribir la lógica de la aplicación. No obstante, este tipo de recuperaciones puede reducir en gran medida el efecto beneficioso para el desempeño que aporta el uso de un índice secundario local.

Cuando elija los atributos para proyectarlos en un índice secundario local, debe estudiar el equilibrio entre los costes de rendimiento aprovisionado y de almacenamiento:

- Si solo necesita acceder a algunos atributos con la menor latencia posible, puede ser conveniente proyectar solamente esos atributos en un índice secundario local. Cuando menor es el índice, menos cuesta almacenarlo y menos son los costes de escritura. Si hay atributos que solo tiene que recuperar de vez en cuando, el costo del desempeño provisionado podría superar con creces el costo a largo plazo de almacenar esos atributos.
- Si la aplicación va a obtener acceso con frecuencia a determinados atributos sin clave, puede ser interesante proyectarlos en un índice secundario local. Los costes del almacenamiento adicionales del índice secundario local compensarán el coste que supondrían los frecuentes exámenes de la tabla.
- Si tiene que acceder a la mayoría de los atributos sin clave con frecuencia, puede proyectar estos atributos o, incluso, toda la tabla base, en un índice secundario local. De este modo, dispondrá de la máxima flexibilidad y el menor consumo de desempeño provisionado, porque no será preciso realizar recuperaciones. Sin embargo, el coste del almacenamiento aumentaría y podría llegar a duplicarse si se proyectan todos los atributos.
- Si la aplicación tiene que consultar una tabla con poca frecuencia, pero tiene que realizar gran cantidad de escrituras o actualizaciones en los datos de la tabla, puede ser conveniente proyectar KEYS_ONLY. El índice secundario local tendrá el tamaño mínimo, pero estaría disponible siempre que se requiriese para actividades de consulta.

Creación de un índice secundario local

Para crear una tabla con uno o varios índices secundarios locales, use el parámetro `LocalSecondaryIndexes` de la operación `CreateTable`. Los índices secundarios locales de una tabla se crean cuando se crea la tabla. Al eliminar una tabla, todos los índices secundarios globales de esa tabla se eliminan también.

Debe especificar un atributo sin clave que actúe como clave de ordenación del índice secundario local. El atributo que elija debe ser un escalar `String`, `Number` o `Binary`. No se admite ningún otro tipo de escalar, documento ni conjunto. Para obtener una lista completa de los tipos de datos, consulte Tipos de datos.

> **Important**
>
> Para las tablas con índices secundarios locales, existe un límite de tamaño de 10 GB por valor de clave de partición. Una tabla con índices secundarios locales puede almacenar cualquier número de elementos, siempre y cuando el tamaño total de cualquier valor de clave de partición individual no supere los 10 GB. Para obtener más información, consulte Límite del tamaño de una colección de elementos.

Puede proyectar atributos de cualquier tipo de datos en un índice secundario local. Esto incluye escalares, documentos y conjuntos. Para obtener una lista completa de los tipos de datos, consulte Tipos de datos.

Lectura de datos de un índice secundario local

Puede recuperar elementos de un índice secundario local utilizando las operaciones `Query` y `Scan`. Las operaciones `GetItem` y `BatchGetItem` no se pueden usar en un índice secundario local.

Consulta a un índice secundario local

En una tabla de DynamoDB, la combinación de valor de clave de partición y valor de clave de ordenación de cada elemento debe ser única. Sin embargo, en un índice secundario local, no es preciso que el valor de la clave de ordenación sea exclusivo para un determinado valor de clave de partición. Si un índice secundario local contiene varios elementos que tienen el mismo valor de clave de ordenación, una operación `Query` devolverá todos los elementos que tengan el mismo valor de clave de partición. En la respuesta, los elementos coincidentes no se devuelven en ningún orden concreto.

Puede consultar un índice secundario local usando lecturas consistentes finales o de consistencia alta. Para especificar qué tipo de consistencia desea aplicar, se utiliza el parámetro `ConsistentReadConsistentead` de la operación `Query`. Una lectura de consistencia alta de un índice secundario local siempre devolverá los valores actualizados más recientemente. Si la consulta tiene que recuperar atributos adicionales de la tabla base, estos serán consistentes respecto al índice.

Example

Fíjese en los datos siguientes devueltos por una operación `Query` que solicita datos de las conversaciones de un foro determinado.

```
{
    "TableName": "Thread",
    "IndexName": "LastPostIndex",
    "ConsistentRead": false,
    "ProjectionExpression": "Subject, LastPostDateTime, Replies, Tags",
    "KeyConditionExpression":
        "ForumName = :v_forum and LastPostDateTime between :v_start and :v_end",
    "ExpressionAttributeValues": {
        ":v_start": {"S": "2015-08-31T00:00:00.000Z"},
        ":v_end": {"S": "2015-11-31T00:00:00.000Z"},
        ":v_forum": {"S": "EC2"}
    }
}
```

En esta consulta:

- DynamoDB accede a `LastPostIndex`. Utiliza la clave de partición `ForumName` para localizar los elementos del índice correspondientes a "EC2". Todos los elementos de índice que tienen esta clave se almacenan en posiciones adyacentes, para agilizar su recuperación.
- En este foro, DynamoDB utiliza el índice para buscar las claves que coinciden con la condición `LastPostDateTime` especificada.
- Dado que el atributo `Replies` se proyecta en el índice, DynamoDB puede recuperar este atributo sin consumir ningún rendimiento provisionado adicional.
- El atributo `Tags` no se proyecta en el índice, de tal forma que DynamoDB debe acceder a la tabla `Thread` y recuperar este atributo.
- Se devuelven los resultados ordenados según `LastPostDateTime`. Las entradas de índice se ordenan según el valor de la clave de partición y, a continuación, según el valor de la clave de

ordenación; `Query` las devuelve en el orden en el que se encuentran almacenadas. Puede utilizar el parámetro `ScanIndexForward` para devolver los resultados en orden descendente.

Puesto que el atributo `Tags` no se ha proyectado en el índice secundario local, DynamoDB debe consumir unidades de capacidad de lectura adicionales para recuperar este atributo de la tabla base. Si necesita ejecutar esta consulta a menudo, debe proyectar `Tags` en `LastPostIndex` para evitar la obtención de la tabla base. Sin embargo, si necesitaba acceder a `Tags` sólo ocasionalmente, el coste de almacenamiento adicional para proyectar `Tags` en el índice puede no valer la pena.

Análisis de un índice secundario local

Puede utilizar `Scan` para recuperar todos los datos de un índice secundario local. Debe proporcionar el nombre de la tabla base y el nombre del índice en la solicitud. Con una operación `Scan`, DynamoDB lee todos los datos del índice y los devuelve a la aplicación. También puede solicitar que solo se devuelvan algunos de los datos y se descarten los demás. Para ello, se utiliza el parámetro `FilterExpression` del API `Scan`. Para obtener más información, consulte Expresiones de filtro para el análisis.

Escrituras de elementos e índices secundarios locales

DynamoDB mantiene sincronizados automáticamente todos los índices secundarios locales con sus tablas base respectivas. Las aplicaciones nunca escriben directamente en un índice. Sin embargo, es importante que comprenda las implicaciones de cómo DynamoDB mantiene estos índices.

Al crear un índice secundario local, se especifica un atributo que actuará como clave de ordenación del índice. También se puede especificar el tipo de datos de ese atributo. Esto significa que, cada vez que se escribe un elemento en la tabla base, si el elemento define un atributo de clave de índice, su tipo debe coincidir con el tipo de datos del esquema de claves de índice. En el caso de `LastPostIndex`, el tipo de datos de la clave de ordenación `LastPostDateTime` del índice es `String`. Si intenta agregar un elemento a la tabla `Thread`, pero especifica un tipo de datos distinto para `LastPostDateTime` (tal como`Number`), DynamoDB devolverá una excepción `ValidationException`, porque los tipos de datos no coinciden.

No es obligatorio que exista una relación biunívoca entre los elementos de una tabla base y los elementos en un índice secundario local. De hecho, este comportamiento puede ser ventajoso para muchas aplicaciones.

Una tabla con muchos índices secundarios locales devengará costes más elevados por la actividad de escritura que las tablas con menos índices. Para obtener más información, consulte Consideraciones sobre el rendimiento aprovisionado para los índices secundarios locales.

> **Important**
>
> Para las tablas con índices secundarios locales, existe un límite de tamaño de 10 GB por valor de clave de partición. Una tabla con índices secundarios locales puede almacenar cualquier número de elementos, siempre y cuando el tamaño total de cualquier valor de clave de partición individual no supere los 10 GB. Para obtener más información, consulte Límite del tamaño de una colección de elementos.

Consideraciones sobre el rendimiento aprovisionado para los índices secundarios locales

Al crear una tabla en DynamoDB, se provisionan unidades de capacidad de lectura y escritura para la carga de trabajo prevista de esa tabla. Esta carga de trabajo incluye la actividad de lectura y escritura en los índices secundarios locales de la tabla.

Para ver las tarifas vigentes de la capacidad de rendimiento aprovisionada, visite Precios de Amazon DynamoDB.

Unidades de capacidad de lectura

Cuando se realiza una consulta en un índice secundario local, el número de unidades de capacidad de lectura consumidas depende de cómo se obtiene acceso a los datos.

Al igual que ocurre con las consultas de tablas, las consultas de índices pueden utilizar lecturas consistentes finales o de consistencia alta, según el valor de `ConsistentRead`. Una lectura de consistencia alta consume una unidad de capacidad de lectura; una lectura eventualmente consistente consume solo la mitad. Por lo tanto, elegir la opción de lecturas consistentes finales permite reducir los cargos de unidades de capacidad de lectura.

Para las consultas de índices que solo solicitan claves de índice y atributos proyectados, DynamoDB calcula la actividad de lectura provisionada de la misma forma que para las consultas de tablas. La única diferencia es que el cálculo se basa en el tamaño de las entradas del índice, no en el tamaño del elemento en la tabla base. El número de unidades de capacidad de lectura es la suma de todos los tamaños de los atributos proyectados para todos los elementos devueltos; el resultado

se redondea al múltiplo de 4 KB inmediatamente superior. Para obtener más información sobre cómo calcula DynamoDB el consumo de rendimiento aprovisionado, consulte Modo de capacidad aprovisionada de DynamoDB.

Para indexar consultas que leen atributos no proyectados en el índice secundario, DynamoDB tendrá que recuperar esos atributos de la tabla base, además de leer los atributos proyectados en el índice. Estas recuperaciones tienen lugar cuando se incluyen atributos no proyectados en los parámetros `Select` o `ProjectionExpression` de la operación `Query`. Las recuperaciones provocan latencia adicional en las respuestas a las consultas y también devengan un mayor coste de desempeño provisionado. Esto se debe a que, además de las lecturas en el índice secundario local que se han descrito anteriormente, se le cobrarán unidades de capacidad de lectura por cada elemento recuperado de la tabla base. Este cargo incluye la lectura de cada elemento completo de la tabla, no solo de los atributos solicitados.

El tamaño máximo de los resultados que devuelve una operación `Query` es de 1 MB. Esta cifra incluye los tamaños de todos los nombres y valores de los atributos de todos los elementos devueltos. Sin embargo, si una consulta de un índice secundario local hace que DynamoDB recupere atributos de elementos de la tabla base, el tamaño máximo de los datos de los resultados podría ser inferior. En este caso, el tamaño del resultado es la suma de:

- El tamaño de los elementos coincidentes del índice, redondeado al múltiplo de 4 KB inmediatamente superior.
- El tamaño de cada elemento coincidente de la tabla base, redondeado individualmente al múltiplo de 4 KB inmediatamente superior.

Usando esta fórmula, el tamaño máximo de los resultados que devuelve una operación Query también es de 1 MB.

Por ejemplo, tomemos una tabla en la que el tamaño de cada elemento es de 300 bytes. Existe un índice secundario local en esa tabla, pero solo se proyectan en él 200 bytes de cada elemento. Ahora, supongamos que se ejecuta `Query` en este índice, que la consulta requiere recuperaciones en la tabla para cada elemento y que la consulta devuelve 4 elementos. DynamoDB calcula lo siguiente:

- El tamaño de los elementos coincidentes en el índice: 200 bytes × 4 elementos = 800 bytes; este valor se redondea hasta 4 KB.
- El tamaño de cada elemento coincidente de la tabla base: (300 bytes, redondeados al múltiplo de 4 KB inmediatamente superior) × 4 elementos = 16 KB.

Por lo tanto, el tamaño total de los datos del resultado es de 20 KB.

Unidades de capacidad de escritura

Cuando se agrega, actualiza o elimina un elemento de una tabla, la actualización de los índices secundarios locales consumirá unidades de capacidad de escritura provisionadas para la tabla. El coste total de rendimiento aprovisionado de una escritura es la suma de las unidades de capacidad de escritura consumidas al escribir en la tabla y aquellas consumidas al actualizar los índices secundarios locales.

El coste de escribir un elemento en un índice secundario local depende de varios factores:

- Si escribe un nuevo elemento en la tabla que define un atributo indexado o actualiza un elemento existente para definir un atributo indexado no definido previamente, se requiere una operación de escritura para colocar el elemento en el índice.
- Si al actualizar la tabla se cambia el valor de un atributo de clave indexado (de A a B), se requieren dos escrituras: una para eliminar el elemento anterior del índice y otra para colocar el nuevo elemento en el índice.
- Si ya hay un elemento en el índice, pero al escribir en la tabla se elimina el atributo indexado, se requiere una escritura para eliminar la proyección del elemento anterior del índice.
- Si no hay ningún elemento presente en el índice antes o después de actualizar el elemento, no se devenga ningún coste de escritura adicional para el índice.

Al calcular las unidades de capacidad de escritura, en todos estos factores se presupone que el tamaño de cada elemento del índice es menor o igual que el tamaño de elemento de 1 KB. Las entradas de índice de mayor tamaño requieren más unidades de capacidad de escritura. Para minimizar los costes de escritura, es conveniente estudiar qué atributos tendrán que devolver las consultas y proyectar solamente esos atributos en el índice.

Consideraciones sobre el almacenamiento para los índices secundarios locales

Cuando una aplicación escribe un elemento en una tabla, DynamoDB copia automáticamente el subconjunto de atributos correcto en todos los índices secundarios locales en los que deban aparecer esos atributos. Se aplica un cargo en su cuenta de AWS por el almacenamiento del elemento en la tabla base y también por el almacenamiento de los atributos en todos los índices secundarios locales de esa tabla.

La cantidad de espacio utilizado por un elemento de índice es la suma de lo siguiente:

- El tamaño en bytes de la clave principal (clave de partición y clave de ordenación) de la tabla base
- El tamaño en bytes del atributo de clave de índice
- El tamaño en bytes de los atributos proyectados (si procede)
- 100 bytes de gastos generales por cada elemento de índice

Para calcular los requisitos de almacenamiento de un índice secundario local, puede calcular el tamaño medio de un elemento del índice y, a continuación, multiplicarlo por el número de elementos del índice.

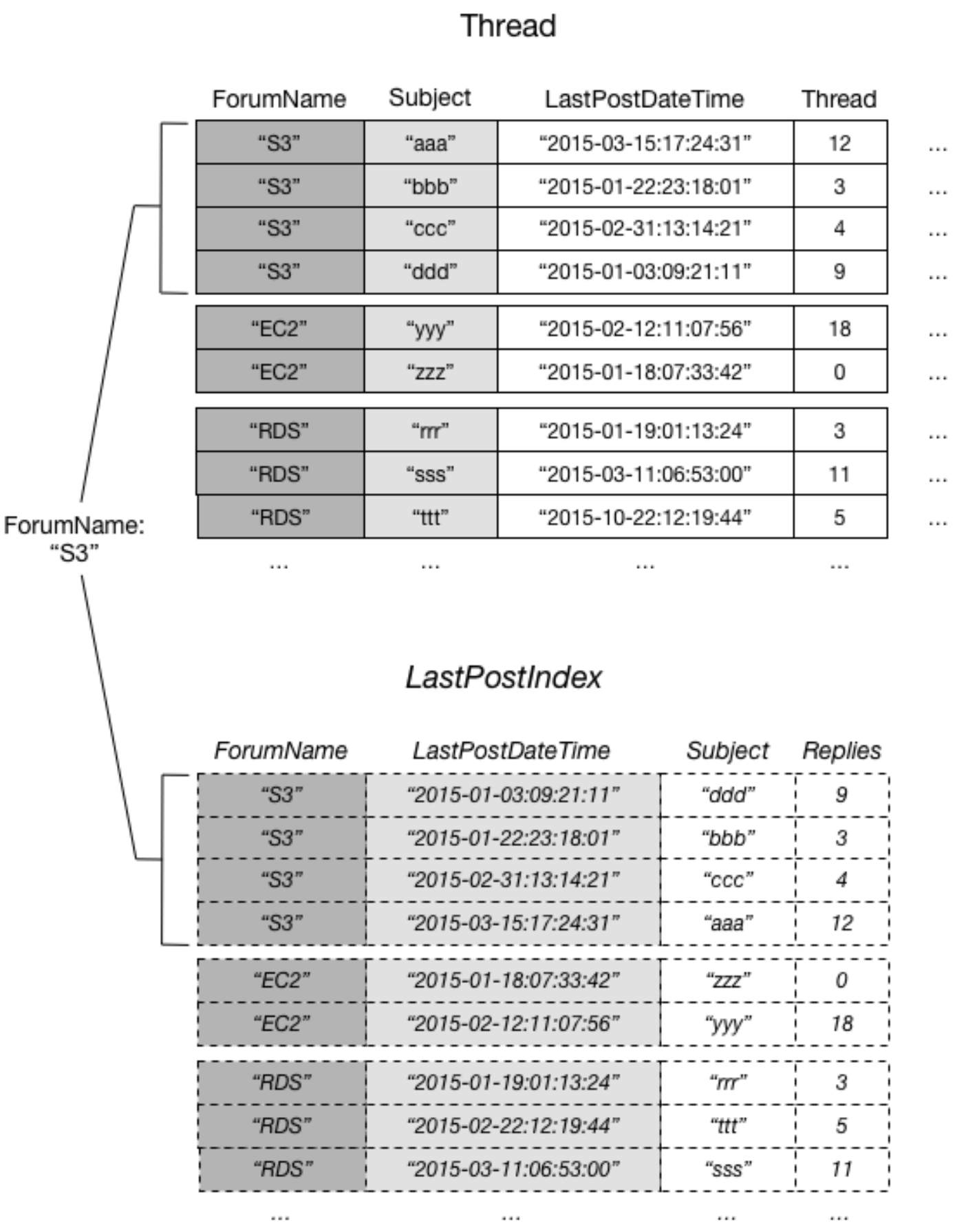
Si una tabla contiene un elemento en el que no se ha definido un atributo determinado, pero ese atributo se ha definido como clave de ordenación del índice, DynamoDB no escribirá ningún dato para ese elemento en el índice.

Colecciones de elementos en los índices secundarios locales

> **Note**
>
> Esta sección solo concierne a las tablas que tienen índices secundarios locales.

En DynamoDB, una colección de elementos es cualquier grupo de elementos que contiene el mismo valor de clave de partición en una tabla y todos sus índices secundarios locales. En los ejemplos utilizados en esta sección, la clave de partición de la tabla `Thread` es `ForumName` y la clave de partición de `LastPostIndex`también es `ForumName`. Todos los elementos de la tabla y del índice que tienen el mismo valor de `ForumName` forman parte de la misma colección de elementos. Por ejemplo, en la tabla `Thread` y en el índice secundario local`LastPostIndex` hay una colección de elementos para el foro `EC2` y otra colección de elementos distinta para el foro `RDS`.

En el siguiente diagrama se muestra la colección de elementos del foro `S3`.


Thread
ForumName
Subject
LastPostDateTime
Thread
"S3"
"aaa"
"2015-03-15:17:24:31"
12
"S3"
"bbb"
"2015-01-22:23:18:01"
3
"S3"
"ccc"
"2015-02-31:13:14:21"
4
"S3"
"ddd"
"2015-01-03:09:21:11"
9
"EC2"
"yyy"
"2015-02-12:11:07:56"
18
"EC2"
"zzz"
"2015-01-18:07:33:42"
0
"RDS"
"rrr"
"2015-01-19:01:13:24"
3
"RDS"
"sss"
"2015-03-11:06:53:00"
11
"RDS"
"ttt"
"2015-10-22:12:19:44"
5
ForumName:
"S3"
LastPostIndex
ForumName
LastPostDateTime
Subject
Replies
"S3"
"2015-01-03:09:21:11"
"ddd"
9
"S3"
"2015-01-22:23:18:01"
"bbb"
3
"S3"
"2015-02-31:13:14:21"
"ccc"
4
"S3"
"2015-03-15:17:24:31"
"aaa"
12
"EC2"
"2015-01-18:07:33:42"
"zzz"
0
"EC2"
"2015-02-12:11:07:56"
"yyy"
18
"RDS"
"2015-01-19:01:13:24"
"rrr"
3
"RDS"
"2015-02-22:12:19:44"
"ttt"
5
"RDS"
"2015-03-11:06:53:00"
"sss"
11

En este diagrama, la colección de elementos consta de todos los elementos de `Thread` y `LastPostIndex` donde el valor de clave de partición `ForumName` es "S3". Si hay otros índices secundarios locales en la tabla, los elementos de esos índices cuyo valor de `ForumName` sea "S3" también formarán parte de la colección de elementos.

Puede utilizar cualquiera de las siguientes operaciones de DynamoDB para devolver información sobre las colecciones de elementos:

- `BatchWriteItem`
- `DeleteItem`
- `PutItem`
- `UpdateItem`
- `TransactWriteItems`

Todas estas operaciones admiten el parámetro `ReturnItemCollectionMetrics`. Si establece este parámetro en `SIZE`, podrá ver información sobre el tamaño de cada colección de elementos en el índice.

Example

A continuación se muestra un ejemplo del resultado de una operación `UpdateItem` en la tabla `Thread`, con `ReturnItemCollectionMetrics` configurado en `SIZE`. El elemento que se ha actualizado tenía un valor de `ForumName` de "EC2", por lo que el resultado incluye información acerca de esa colección de elementos.

```
{
    ItemCollectionMetrics: {
        ItemCollectionKey: {
            ForumName: "EC2"
        },
        SizeEstimateRangeGB: [0.0, 1.0]
    }
}
```

El objeto `SizeEstimateRangeGB` muestra que el tamaño de esta colección de elementos está comprendido entre 0 y 1 GB. DynamoDB actualiza periódicamente este cálculo del tamaño, de modo que las cifras podrían ser distintas la próxima vez que se modifique el elemento.

Límite del tamaño de una colección de elementos

El tamaño máximo de cualquier colección de elementos para una tabla que tenga uno o más índices secundarios locales es de 10 GB. Esto no se aplica a las colecciones de elementos en tablas sin índices secundarios locales, y tampoco se aplica a las colecciones de elementos en índices secundarios generales. Solo se ven afectadas las tablas que tienen uno o más índices secundarios locales.

Si una colección de elementos supera el límite de 10 GB, DynamoDB devolverá una `ItemCollectionSizeLimitExceededException` y ya no podrá agregar más elementos a la colección de elementos ni incrementar los tamaños de los elementos contenidos en ella. (Las operaciones de lectura y escritura que reduzcan el tamaño de la colección de elementos sí se permitirán). También podrá agregar elementos a otras colecciones de elementos.

Para reducir el tamaño de una colección de elementos, puede elegir una de las siguientes opciones:

- Eliminar todos los elementos innecesarios que tengan el valor de clave de partición de que se trate. Al eliminar estos elementos de la tabla base, DynamoDB también eliminará las entradas de índice cuyo valor de clave de partición sea el mismo.
- Actualizar los elementos eliminando atributos o reduciendo el tamaño de estos últimos. Si estos atributos se han proyectado en uno o varios índices secundarios locales, DynamoDB también reducirá el tamaño de las entradas de índice correspondientes.
- Crear una nueva tabla con las mismas clave de partición y clave de ordenación y, a continuación, mover elementos de la tabla anterior a la nueva. Esto puede ser conveniente si una tabla contiene datos históricos a los que se obtiene acceso con poca frecuencia. Puede considerar también el archivado de estos datos históricos en Amazon Simple Storage Service (Amazon S3).

Cuando el tamaño total de la colección de elementos disminuye por debajo de 10 GB, podrá volver a agregar elementos con el mismo valor de clave de partición.

Como práctica recomendada, es preferible instrumentar la aplicación de tal forma que monitorice los tamaños de las colecciones de elementos. Una forma de hacerlo es establecer el parámetro `ReturnItemCollectionMetrics` en `SIZE` siempre que utilice `BatchWriteItem`, `DeleteItem`, `PutItem` o `UpdateItem`. La aplicación debe examinar el objeto `ReturnItemCollectionMetrics` en los resultados y registrar un mensaje de error cada vez que una colección de elementos supere un límite definido por el usuario (8 GB, por ejemplo). Establecer un límite menor que 10 GB ofrece un mecanismo de advertencia precoz sistema que le permitirá

saber qué colección de elementos está próxima a alcanzar el límite a tiempo para adoptar las medidas pertinentes.

Colecciones de elementos y particiones

En una tabla con uno o más índices secundarios locales, cada colección de elementos se almacena en una partición. El tamaño total de esa colección de elementos está limitado a la capacidad de esa partición: 10 GB. En el caso de una aplicación en la que el modelo de datos incluya colecciones de elementos cuyo tamaño no esté limitado, o en la que pueda esperar razonablemente que algunas colecciones de elementos aumenten más allá de los 10 GB en el futuro, debería considerar la posibilidad de utilizar un índice secundario general.

Debe diseñar las aplicaciones de tal forma que los datos de las tablas se distribuyan uniformemente entre los distintos valores de clave de partición. Para las tablas que tienen índices secundarios locales, las aplicaciones no deben crear "puntos calientes" de actividad de lectura y escritura dentro de una misma colección de elementos de una sola partición.

Trabajar con índices secundarios locales: Java

Puede utilizar la API de documentos AWS SDK para Java para crear una tabla Amazon DynamoDB con uno o varios índices secundarios locales, describir los índices de la tabla y utilizarlos para realizar consultas.

A continuación se indican los pasos comunes para las operaciones con tablas mediante el API de documentos del AWS SDK para Java.

1. Cree una instancia de la clase `DynamoDB`.
2. Cree los objetos de solicitud correspondientes para proporcionar los parámetros obligatorios y opcionales de la operación.
3. Llame al método apropiado proporcionado por el cliente que ha creado en el paso anterior.

Temas

- Creación de una tabla con un índice secundario local
- Descripción de una tabla con un índice secundario local
- Consulta de un índice secundario local
- Ejemplo: índices secundarios locales mediante la API de documentos de Java

Creación de una tabla con un índice secundario local

Los índices secundarios locales se deben crear a la vez que se crea la tabla. Para ello, utilice el método `createTable` e indique las especificaciones para uno o varios índices secundarios locales. En el ejemplo de código Java siguiente, se crea una tabla para contener información sobre las canciones de una colección de música. La clave de partición es `Artist` y la de ordenación, `SongTitle`. Un índice secundario, `AlbumTitleIndex`, facilita las consultas por título de álbum.

A continuación se indican los pasos que hay que seguir para crear una tabla con un índice secundario local mediante la API de documentos de DynamoDB.

1. Cree una instancia de la clase `DynamoDB`.
2. Cree una instancia de la clase `CreateTableRequest` para proporcionar la información de solicitud.

   Debe proporcionar el nombre de la tabla, su clave principal y los valores de rendimiento aprovisionado. Para el índice secundario local, debe proporcionar el nombre de índice, el nombre y el tipo de datos de la clave de ordenación del índice, el esquema de claves del índice y la proyección de atributos.
3. Llame al método `createTable` proporcionando el objeto de solicitud como parámetro.

En el siguiente ejemplo de código Java se muestran los pasos anteriores. En el fragmento se crea una tabla (`Music`) con un índice secundario basado en el atributo `AlbumTitle`. La clave de partición y la clave de ordenación de la tabla, además de la clave de ordenación del índice, son los únicos atributos proyectados en el índice.

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";

CreateTableRequest createTableRequest = new
 CreateTableRequest().withTableName(tableName);

//ProvisionedThroughput
createTableRequest.setProvisionedThroughput(new
 ProvisionedThroughput().withReadCapacityUnits((long)5).withWriteCapacityUnits((long)5));

//AttributeDefinitions
```

```
ArrayList<AttributeDefinition> attributeDefinitions= new
 ArrayList<AttributeDefinition>();
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Artist").withAttributeType("S"));
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("SongTitle").withAttributeType("S"));
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("AlbumTitle").withAttributeType("S"));

createTableRequest.setAttributeDefinitions(attributeDefinitions);

//KeySchema
ArrayList<KeySchemaElement> tableKeySchema = new ArrayList<KeySchemaElement>();
tableKeySchema.add(new
 KeySchemaElement().withAttributeName("Artist").withKeyType(KeyType.HASH));  //
Partition key
tableKeySchema.add(new
 KeySchemaElement().withAttributeName("SongTitle").withKeyType(KeyType.RANGE));  //Sort
 key

createTableRequest.setKeySchema(tableKeySchema);

ArrayList<KeySchemaElement> indexKeySchema = new ArrayList<KeySchemaElement>();
indexKeySchema.add(new
 KeySchemaElement().withAttributeName("Artist").withKeyType(KeyType.HASH));  //
Partition key
indexKeySchema.add(new
 KeySchemaElement().withAttributeName("AlbumTitle").withKeyType(KeyType.RANGE));  //
Sort key

Projection projection = new Projection().withProjectionType(ProjectionType.INCLUDE);
ArrayList<String> nonKeyAttributes = new ArrayList<String>();
nonKeyAttributes.add("Genre");
nonKeyAttributes.add("Year");
projection.setNonKeyAttributes(nonKeyAttributes);

LocalSecondaryIndex localSecondaryIndex = new LocalSecondaryIndex()

 .withIndexName("AlbumTitleIndex").withKeySchema(indexKeySchema).withProjection(projection);

ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
 ArrayList<LocalSecondaryIndex>();
localSecondaryIndexes.add(localSecondaryIndex);
createTableRequest.setLocalSecondaryIndexes(localSecondaryIndexes);
```

```
Table table = dynamoDB.createTable(createTableRequest);
System.out.println(table.getDescription());
```

Debe esperar hasta que DynamoDB cree la tabla y establezca el estado de esta última en `ACTIVE`. A partir de ese momento, puede comenzar a incluir elementos de datos en la tabla.

Descripción de una tabla con un índice secundario local

Para obtener información acerca de los índices secundarios locales en una tabla, utilice el método `describeTable`. Para cada índice, puede obtener acceso a su nombre, esquema de claves y atributos proyectados.

A continuación, se muestran los pasos que hay que seguir para obtener acceso a la información de un índice secundario local de una tabla mediante la API de documentos de AWS SDK para Java.

1. Cree una instancia de la clase `DynamoDB`.
2. Cree una instancia de la clase `Table`. Debe proporcionar el nombre de la tabla.
3. Llame al método `describeTable` del objeto `Table`.

En el siguiente ejemplo de código Java se muestran los pasos anteriores.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";

Table table = dynamoDB.getTable(tableName);

TableDescription tableDescription = table.describe();

List<LocalSecondaryIndexDescription> localSecondaryIndexes
    = tableDescription.getLocalSecondaryIndexes();

// This code snippet will work for multiple indexes, even though
// there is only one index in this example.

Iterator<LocalSecondaryIndexDescription> lsiIter = localSecondaryIndexes.iterator();
while (lsiIter.hasNext()) {
```

```
    LocalSecondaryIndexDescription lsiDescription = lsiIter.next();
    System.out.println("Info for index " + lsiDescription.getIndexName() + ":");
    Iterator<KeySchemaElement> kseIter = lsiDescription.getKeySchema().iterator();
    while (kseIter.hasNext()) {
        KeySchemaElement kse = kseIter.next();
        System.out.printf("\t%s: %s\n", kse.getAttributeName(), kse.getKeyType());
    }
    Projection projection = lsiDescription.getProjection();
    System.out.println("\tThe projection type is: " + projection.getProjectionType());
    if (projection.getProjectionType().toString().equals("INCLUDE")) {
        System.out.println("\t\tThe non-key projected attributes are: " +
 projection.getNonKeyAttributes());
    }
}
```

Consulta de un índice secundario local

Puede utilizar la operación `Query` en un índice secundario local de un modo bastante parecido a como `Query` se usa en una tabla. Debe especificar el nombre del índice, los criterios de consulta de la clave de ordenación del índice y los atributos que desea devolver. En este ejemplo, el índice es `AlbumTitleIndex` y la clave de ordenación del índice es `AlbumTitle`.

Los únicos atributos devueltos son aquellos que se han proyectado en el índice. Puede modificar esta consulta de modo que también seleccione atributos sin clave, pero esto requeriría realizar actividad de recuperación en la tabla, lo que resulta relativamente costoso. Para obtener más información sobre recuperaciones de tablas, consulte Proyecciones de atributos.

A continuación se indican los pasos que hay que seguir para consultar un índice secundario local con la API de documentos de AWS SDK para Java.

1. Cree una instancia de la clase `DynamoDB`.
2. Cree una instancia de la clase `Table`. Debe proporcionar el nombre de la tabla.
3. Cree una instancia de la clase `Index`. Debe proporcionar el nombre del índice.
4. Llame al método `query` de la clase `Index`.

En el siguiente ejemplo de código Java se muestran los pasos anteriores.

Example

```
AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
DynamoDB dynamoDB = new DynamoDB(client);

String tableName = "Music";

Table table = dynamoDB.getTable(tableName);
Index index = table.getIndex("AlbumTitleIndex");

QuerySpec spec = new QuerySpec()
    .withKeyConditionExpression("Artist = :v_artist and AlbumTitle = :v_title")
    .withValueMap(new ValueMap()
        .withString(":v_artist", "Acme Band")
        .withString(":v_title", "Songs About Life"));

ItemCollection<QueryOutcome> items = index.query(spec);

Iterator<Item> itemsIter = items.iterator();

while (itemsIter.hasNext()) {
    Item item = itemsIter.next();
    System.out.println(item.toJSONPretty());
}
```

Ejemplo: índices secundarios locales mediante la API de documentos de Java

En el siguiente ejemplo de código Java se muestra cómo usar índices secundarios locales en Amazon DynamoDB. En el ejemplo se crea una tabla denominada `CustomerOrders` cuya clave de partición es `CustomerId` y cuya clave de ordenación es `OrderId`. Hay dos índices secundarios locales en esta tabla:

- `OrderCreationDateIndex`: la clave de ordenación es `OrderCreationDate` y los atributos siguientes se proyectan en el índice:
  - `ProductCategory`
  - `ProductName`
  - `OrderStatus`
  - `ShipmentTrackingId`
- `IsOpenIndex`: la clave de ordenación es `IsOpen` y todos los atributos de la tabla se proyectan en el índice.

Después de que se crea la tabla `CustomerOrders`, el programa carga la tabla con datos que representan pedidos de clientes. A continuación, consulta los datos utilizando los índices secundarios locales. Por último, el programa elimina la tabla `CustomerOrders`.

Para obtener instrucciones paso a paso para probar el siguiente ejemplo, consulte Ejemplos de código Java.

Example

```
package com.amazonaws.codesamples.document;

import java.util.ArrayList;
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Index;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.PutItemOutcome;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;
import com.amazonaws.services.dynamodbv2.document.utils.ValueMap;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.LocalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProjectionType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ReturnConsumedCapacity;
import com.amazonaws.services.dynamodbv2.model.Select;

public class DocumentAPILocalSecondaryIndexExample {

        static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
        static DynamoDB dynamoDB = new DynamoDB(client);

        public static String tableName = "CustomerOrders";
```

```
        public static void main(String[] args) throws Exception {

                createTable();
                loadData();

                query(null);
                query("IsOpenIndex");
                query("OrderCreationDateIndex");

                deleteTable(tableName);

        }

        public static void createTable() {

                CreateTableRequest createTableRequest = new
CreateTableRequest().withTableName(tableName)
                                .withProvisionedThroughput(
                                                new
ProvisionedThroughput().withReadCapacityUnits((long) 1)

.withWriteCapacityUnits((long) 1));

                // Attribute definitions for table partition and sort keys
                ArrayList<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();
                attributeDefinitions
                                .add(new
AttributeDefinition().withAttributeName("CustomerId").withAttributeType("S"));
                attributeDefinitions.add(new
AttributeDefinition().withAttributeName("OrderId").withAttributeType("N"));

                // Attribute definition for index primary key attributes
                attributeDefinitions
                                .add(new
AttributeDefinition().withAttributeName("OrderCreationDate")
                                                .withAttributeType("N"));
                attributeDefinitions.add(new
AttributeDefinition().withAttributeName("IsOpen").withAttributeType("N"));

                createTableRequest.setAttributeDefinitions(attributeDefinitions);

                // Key schema for table
```

```
                ArrayList<KeySchemaElement> tableKeySchema = new
ArrayList<KeySchemaElement>();
                tableKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition

                                  // key
                tableKeySchema.add(new
KeySchemaElement().withAttributeName("OrderId").withKeyType(KeyType.RANGE)); // Sort

                                 // key

                createTableRequest.setKeySchema(tableKeySchema);

                ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
ArrayList<LocalSecondaryIndex>();

                // OrderCreationDateIndex
                LocalSecondaryIndex orderCreationDateIndex = new LocalSecondaryIndex()
                                .withIndexName("OrderCreationDateIndex");

                // Key schema for OrderCreationDateIndex
                ArrayList<KeySchemaElement> indexKeySchema = new
ArrayList<KeySchemaElement>();
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition

                                // key
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("OrderCreationDate")
                                .withKeyType(KeyType.RANGE)); // Sort
                                                                // key

                orderCreationDateIndex.setKeySchema(indexKeySchema);

                // Projection (with list of projected attributes) for
                // OrderCreationDateIndex
                Projection projection = new
Projection().withProjectionType(ProjectionType.INCLUDE);
                ArrayList<String> nonKeyAttributes = new ArrayList<String>();
                nonKeyAttributes.add("ProductCategory");
                nonKeyAttributes.add("ProductName");
                projection.setNonKeyAttributes(nonKeyAttributes);
```

```
                orderCreationDateIndex.setProjection(projection);

                localSecondaryIndexes.add(orderCreationDateIndex);

                // IsOpenIndex
                LocalSecondaryIndex isOpenIndex = new
LocalSecondaryIndex().withIndexName("IsOpenIndex");

                // Key schema for IsOpenIndex
                indexKeySchema = new ArrayList<KeySchemaElement>();
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("CustomerId").withKeyType(KeyType.HASH)); //
Partition

                                // key
                indexKeySchema.add(new
KeySchemaElement().withAttributeName("IsOpen").withKeyType(KeyType.RANGE)); // Sort

                            // key

                // Projection (all attributes) for IsOpenIndex
                projection = new Projection().withProjectionType(ProjectionType.ALL);

                isOpenIndex.setKeySchema(indexKeySchema);
                isOpenIndex.setProjection(projection);

                localSecondaryIndexes.add(isOpenIndex);

                // Add index definitions to CreateTable request
                createTableRequest.setLocalSecondaryIndexes(localSecondaryIndexes);

                System.out.println("Creating table " + tableName + "...");
                System.out.println(dynamoDB.createTable(createTableRequest));

                // Wait for table to become active
                System.out.println("Waiting for " + tableName + " to become
ACTIVE...");
                try {
                        Table table = dynamoDB.getTable(tableName);
                        table.waitForActive();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
```

```
        }

        public static void query(String indexName) {

                Table table = dynamoDB.getTable(tableName);


System.out.println("\n***********************************************************\n");
                System.out.println("Querying table " + tableName + "...");

                QuerySpec querySpec = new
QuerySpec().withConsistentRead(true).withScanIndexForward(true)

.withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

                if (indexName == "IsOpenIndex") {

                        System.out.println("\nUsing index: '" + indexName + "': Bob's
orders that are open.");
                        System.out.println("Only a user-specified list of attributes
are returned\n");
                        Index index = table.getIndex(indexName);

                        querySpec.withKeyConditionExpression("CustomerId = :v_custid
and IsOpen = :v_isopen")
                                        .withValueMap(new
ValueMap().withString(":v_custid", "bob@example.com")
                                                        .withNumber(":v_isopen", 1));

                        querySpec.withProjectionExpression(
                                        "OrderCreationDate, ProductCategory,
ProductName, OrderStatus");

                        ItemCollection<QueryOutcome> items = index.query(querySpec);
                        Iterator<Item> iterator = items.iterator();

                        System.out.println("Query: printing results...");

                        while (iterator.hasNext()) {
                                System.out.println(iterator.next().toJSONPretty());
                        }

                } else if (indexName == "OrderCreationDateIndex") {
                        System.out.println("\nUsing index: '" + indexName
```

```
                                        + "': Bob's orders that were placed after
 01/31/2015.");
                        System.out.println("Only the projected attributes are returned
\n");
                        Index index = table.getIndex(indexName);

                        querySpec.withKeyConditionExpression(
                                        "CustomerId = :v_custid and OrderCreationDate
 >= :v_orddate")
                                        .withValueMap(
                                                        new
 ValueMap().withString(":v_custid", "bob@example.com")

 .withNumber(":v_orddate",

 20150131));

                        querySpec.withSelect(Select.ALL_PROJECTED_ATTRIBUTES);

                        ItemCollection<QueryOutcome> items = index.query(querySpec);
                        Iterator<Item> iterator = items.iterator();

                        System.out.println("Query: printing results...");

                        while (iterator.hasNext()) {
                                System.out.println(iterator.next().toJSONPretty());
                        }

                } else {
                        System.out.println("\nNo index: All of Bob's orders, by
 OrderId:\n");

                        querySpec.withKeyConditionExpression("CustomerId = :v_custid")
                                        .withValueMap(new
 ValueMap().withString(":v_custid", "bob@example.com"));

                        ItemCollection<QueryOutcome> items = table.query(querySpec);
                        Iterator<Item> iterator = items.iterator();

                        System.out.println("Query: printing results...");

                        while (iterator.hasNext()) {
                                System.out.println(iterator.next().toJSONPretty());
                        }
```

```
                }

        }

        public static void deleteTable(String tableName) {

                Table table = dynamoDB.getTable(tableName);
                System.out.println("Deleting table " + tableName + "...");
                table.delete();

                // Wait for table to be deleted
                System.out.println("Waiting for " + tableName + " to be deleted...");
                try {
                        table.waitForDelete();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }

        public static void loadData() {

                Table table = dynamoDB.getTable(tableName);

                System.out.println("Loading data into table " + tableName + "...");

                Item item = new Item().withPrimaryKey("CustomerId",
"alice@example.com").withNumber("OrderId", 1)
                                .withNumber("IsOpen",
1).withNumber("OrderCreationDate", 20150101)
                                .withString("ProductCategory", "Book")
                                .withString("ProductName", "The Great Outdoors")
                                .withString("OrderStatus", "PACKING ITEMS");
                // no ShipmentTrackingId attribute

                PutItemOutcome putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"alice@example.com").withNumber("OrderId", 2)
                                .withNumber("IsOpen",
1).withNumber("OrderCreationDate", 20150221)
                                .withString("ProductCategory", "Bike")
                                .withString("ProductName", "Super Mountain")
                                .withString("OrderStatus", "ORDER RECEIVED");
```

```
                // no ShipmentTrackingId attribute

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"alice@example.com").withNumber("OrderId", 3)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150304).withString("ProductCategory", "Music")
                                .withString("ProductName", "A Quiet
Interlude").withString("OrderStatus", "IN TRANSIT")
                                .withString("ShipmentTrackingId", "176493");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 1)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150111).withString("ProductCategory", "Movie")
                                .withString("ProductName", "Calm Before The Storm")
                                .withString("OrderStatus", "SHIPPING DELAY")
                                .withString("ShipmentTrackingId", "859323");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 2)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150124).withString("ProductCategory", "Music")
                                .withString("ProductName", "E-Z
Listening").withString("OrderStatus", "DELIVERED")
                                .withString("ShipmentTrackingId", "756943");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 3)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150221).withString("ProductCategory", "Music")
                                .withString("ProductName", "Symphony
9").withString("OrderStatus", "DELIVERED")
```

```
                                .withString("ShipmentTrackingId", "645193");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 4)
                                .withNumber("IsOpen",
1).withNumber("OrderCreationDate", 20150222)
                                .withString("ProductCategory", "Hardware")
                                .withString("ProductName", "Extra Heavy Hammer")
                                .withString("OrderStatus", "PACKING ITEMS");
                // no ShipmentTrackingId attribute

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 5)
                                /* no IsOpen attribute */
                                .withNumber("OrderCreationDate",
20150309).withString("ProductCategory", "Book")
                                .withString("ProductName", "How To
Cook").withString("OrderStatus", "IN TRANSIT")
                                .withString("ShipmentTrackingId", "440185");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 6)
                                // no IsOpen attribute
                                .withNumber("OrderCreationDate",
20150318).withString("ProductCategory", "Luggage")
                                .withString("ProductName", "Really Big
Suitcase").withString("OrderStatus", "DELIVERED")
                                .withString("ShipmentTrackingId", "893927");

                putItemOutcome = table.putItem(item);

                item = new Item().withPrimaryKey("CustomerId",
"bob@example.com").withNumber("OrderId", 7)
                                /* no IsOpen attribute */
                                .withNumber("OrderCreationDate",
20150324).withString("ProductCategory", "Golf")
                                .withString("ProductName", "PGA Pro
II").withString("OrderStatus", "OUT FOR DELIVERY")
```

```
                                .withString("ShipmentTrackingId", "383283");

                putItemOutcome = table.putItem(item);
                assert putItemOutcome != null;
        }

}
```

Trabajar con índices secundarios locales: .NET

Temas

- Creación de una tabla con un índice secundario local
- Descripción de una tabla con un índice secundario local
- Consulta de un índice secundario local
- Ejemplo: índices secundarios locales que utilizan la API de bajo nivel de AWS SDK para .NET

Puede utilizar la API de bajo nivel AWS SDK para .NET para crear una tabla Amazon DynamoDB con uno o varios índices secundarios locales, describir los índices de la tabla y utilizarlos para realizar consultas. Estas operaciones se mapean a las acciones correspondientes del API de bajo nivel de DynamoDB. Para obtener más información, consulte Ejemplos de código .NET.

A continuación se indican los pasos comunes para las operaciones con tablas mediante el API de bajo nivel de .NET.

1. Cree una instancia de la clase `AmazonDynamoDBClient`.
2. Cree los objetos de solicitud correspondientes para proporcionar los parámetros obligatorios y opcionales de la operación.

   Por ejemplo, cree un objeto `CreateTableRequest` para crear una tabla y un objeto `QueryRequest` para consultar una tabla o un índice.
3. Ejecute el método apropiado proporcionado por el cliente que ha creado en el paso anterior.

Creación de una tabla con un índice secundario local

Los índices secundarios locales se deben crear a la vez que se crea la tabla. Para ello, utilice `CreateTable` e indique las especificaciones para uno o varios índices secundarios globales. En el ejemplo de código C# siguiente, se crea una tabla para contener información sobre las canciones

de una colección de música. La clave de partición es `Artist` y la de ordenación, `SongTitle`. Un índice secundario, `AlbumTitleIndex`, facilita las consultas por título de álbum.

A continuación se indican los pasos que hay que seguir para crear una tabla con un índice secundario local mediante la API de bajo nivel de .NET.

1. Cree una instancia de la clase `AmazonDynamoDBClient`.
2. Cree una instancia de la clase `CreateTableRequest` para proporcionar la información de solicitud.

   Debe proporcionar el nombre de la tabla, su clave principal y los valores de rendimiento aprovisionado. Para el índice secundario local, debe proporcionar el nombre de índice, el nombre y el tipo de datos de la clave de ordenación del índice, el esquema de claves del índice y la proyección de atributos.
3. Ejecute el método `CreateTable` proporcionando el objeto de solicitud como parámetro.

En el siguiente ejemplo de código C# se ponen en práctica los pasos anteriores. En el fragmento se crea una tabla (`Music`) con un índice secundario basado en el atributo `AlbumTitle`. La clave de partición y la clave de ordenación de la tabla, además de la clave de ordenación del índice, son los únicos atributos proyectados en el índice.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "Music";

CreateTableRequest createTableRequest = new CreateTableRequest()
{
    TableName = tableName
};

//ProvisionedThroughput
createTableRequest.ProvisionedThroughput = new ProvisionedThroughput()
{
    ReadCapacityUnits = (long)5,
    WriteCapacityUnits = (long)5
};

//AttributeDefinitions
List<AttributeDefinition> attributeDefinitions = new List<AttributeDefinition>();

attributeDefinitions.Add(new AttributeDefinition()
```

```
{
    AttributeName = "Artist",
    AttributeType = "S"
});

attributeDefinitions.Add(new AttributeDefinition()
 {
     AttributeName = "SongTitle",
     AttributeType = "S"
 });

attributeDefinitions.Add(new AttributeDefinition()
 {
     AttributeName = "AlbumTitle",
     AttributeType = "S"
 });

createTableRequest.AttributeDefinitions = attributeDefinitions;

//KeySchema
List<KeySchemaElement> tableKeySchema = new List<KeySchemaElement>();

tableKeySchema.Add(new KeySchemaElement() { AttributeName = "Artist", KeyType =
 "HASH" });  //Partition key
tableKeySchema.Add(new KeySchemaElement() { AttributeName = "SongTitle", KeyType =
 "RANGE" });  //Sort key

createTableRequest.KeySchema = tableKeySchema;

List<KeySchemaElement> indexKeySchema = new List<KeySchemaElement>();
indexKeySchema.Add(new KeySchemaElement() { AttributeName = "Artist", KeyType =
 "HASH" });  //Partition key
indexKeySchema.Add(new KeySchemaElement() { AttributeName = "AlbumTitle", KeyType =
 "RANGE" });  //Sort key

Projection projection = new Projection() { ProjectionType = "INCLUDE" };

List<string> nonKeyAttributes = new List<string>();
nonKeyAttributes.Add("Genre");
nonKeyAttributes.Add("Year");
projection.NonKeyAttributes = nonKeyAttributes;

LocalSecondaryIndex localSecondaryIndex = new LocalSecondaryIndex()
{
```

```
    IndexName = "AlbumTitleIndex",
    KeySchema = indexKeySchema,
    Projection = projection
};

List<LocalSecondaryIndex> localSecondaryIndexes = new List<LocalSecondaryIndex>();
localSecondaryIndexes.Add(localSecondaryIndex);
createTableRequest.LocalSecondaryIndexes = localSecondaryIndexes;

CreateTableResponse result = client.CreateTable(createTableRequest);
Console.WriteLine(result.CreateTableResult.TableDescription.TableName);
Console.WriteLine(result.CreateTableResult.TableDescription.TableStatus);
```

Debe esperar hasta que DynamoDB cree la tabla y establezca el estado de esta última en `ACTIVE`. A partir de ese momento, puede comenzar a incluir elementos de datos en la tabla.

Descripción de una tabla con un índice secundario local

Para obtener información acerca de los índices secundarios locales en una tabla, utilice la API `DescribeTable`. Para cada índice, puede obtener acceso a su nombre, esquema de claves y atributos proyectados.

A continuación se indican los pasos que hay que seguir para acceder a la información de un índice secundario local de una tabla mediante el API de bajo nivel de .NET.

1. Cree una instancia de la clase `AmazonDynamoDBClient`.
2. Cree una instancia de la clase `DescribeTableRequest` para proporcionar la información de solicitud. Debe proporcionar el nombre de la tabla.
3. Ejecute el método `describeTable` proporcionando el objeto de solicitud como parámetro.
4.

En el siguiente ejemplo de código C# se ponen en práctica los pasos anteriores.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "Music";

DescribeTableResponse response = client.DescribeTable(new DescribeTableRequest()
 { TableName = tableName });
List<LocalSecondaryIndexDescription> localSecondaryIndexes =
```

```
    response.DescribeTableResult.Table.LocalSecondaryIndexes;

// This code snippet will work for multiple indexes, even though
// there is only one index in this example.
foreach (LocalSecondaryIndexDescription lsiDescription in localSecondaryIndexes)
{
    Console.WriteLine("Info for index " + lsiDescription.IndexName + ":");

    foreach (KeySchemaElement kse in lsiDescription.KeySchema)
    {
        Console.WriteLine("\t" + kse.AttributeName + ": key type is " + kse.KeyType);
    }

    Projection projection = lsiDescription.Projection;

    Console.WriteLine("\tThe projection type is: " + projection.ProjectionType);

    if (projection.ProjectionType.ToString().Equals("INCLUDE"))
    {
        Console.WriteLine("\t\tThe non-key projected attributes are:");

        foreach (String s in projection.NonKeyAttributes)
        {
            Console.WriteLine("\t\t" + s);
        }

    }
}
```

Consulta de un índice secundario local

Puede utilizar `Query` en un índice secundario local de un modo bastante parecido a como usa `Query` en una tabla. Debe especificar el nombre del índice, los criterios de consulta de la clave de ordenación del índice y los atributos que desea devolver. En este ejemplo, el índice es `AlbumTitleIndex` y la clave de ordenación del índice es `AlbumTitle`.

Los únicos atributos devueltos son aquellos que se han proyectado en el índice. Puede modificar esta consulta de modo que también seleccione atributos sin clave, pero esto requeriría realizar actividad de recuperación en la tabla, lo que resulta relativamente costoso. Para obtener más información sobre recuperaciones de tablas, consulte Proyecciones de atributos.

A continuación se indican los pasos que hay que seguir para consultar un índice secundario local mediante la API de bajo nivel de .NET.

1. Cree una instancia de la clase `AmazonDynamoDBClient`.
2. Cree una instancia de la clase `QueryRequest` para proporcionar la información de solicitud.
3. Ejecute el método `query` proporcionando el objeto de solicitud como parámetro.

En el siguiente ejemplo de código C# se ponen en práctica los pasos anteriores.

Example

```
QueryRequest queryRequest = new QueryRequest
{
    TableName = "Music",
    IndexName = "AlbumTitleIndex",
    Select = "ALL_ATTRIBUTES",
    ScanIndexForward = true,
    KeyConditionExpression = "Artist = :v_artist and AlbumTitle = :v_title",
    ExpressionAttributeValues = new Dictionary<string, AttributeValue>()
    {
        {":v_artist",new AttributeValue {S = "Acme Band"}},
        {":v_title",new AttributeValue {S = "Songs About Life"}}
    },
};

QueryResponse response = client.Query(queryRequest);

foreach (var attribs in response.Items)
{
    foreach (var attrib in attribs)
    {
        Console.WriteLine(attrib.Key + " ---> " + attrib.Value.S);
    }
    Console.WriteLine();
}
```

Ejemplo: índices secundarios locales que utilizan la API de bajo nivel de AWS SDK para .NET

En el siguiente ejemplo de código C# se muestra cómo usar índices secundarios locales en Amazon DynamoDB. En el ejemplo se crea una tabla denominada `CustomerOrders` cuya clave de partición es `CustomerId` y cuya clave de ordenación es `OrderId`. Hay dos índices secundarios locales en esta tabla:

- `OrderCreationDateIndex`: la clave de ordenación es `OrderCreationDate` y los atributos siguientes se proyectan en el índice:
  - `ProductCategory`
  - `ProductName`
  - `OrderStatus`
  - `ShipmentTrackingId`
- `IsOpenIndex`: la clave de ordenación es `IsOpen` y todos los atributos de la tabla se proyectan en el índice.

Después de que se crea la tabla `CustomerOrders`, el programa carga la tabla con datos que representan pedidos de clientes. A continuación, consulta los datos utilizando los índices secundarios locales. Por último, el programa elimina la tabla `CustomerOrders`.

Para obtener instrucciones paso a paso para probar el siguiente ejemplo, consulte Ejemplos de código .NET.

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class LowLevelLocalSecondaryIndexExample
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();
        private static string tableName = "CustomerOrders";

        static void Main(string[] args)
        {
            try
            {
                CreateTable();
```

```
            LoadData();

            Query(null);
            Query("IsOpenIndex");
            Query("OrderCreationDateIndex");

            DeleteTable(tableName);

            Console.WriteLine("To continue, press Enter");
            Console.ReadLine();
        }
        catch (AmazonDynamoDBException e) { Console.WriteLine(e.Message); }
        catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
        catch (Exception e) { Console.WriteLine(e.Message); }
    }

    private static void CreateTable()
    {
        var createTableRequest =
            new CreateTableRequest()
            {
                TableName = tableName,
                ProvisionedThroughput =
                new ProvisionedThroughput()
                {
                    ReadCapacityUnits = (long)1,
                    WriteCapacityUnits = (long)1
                }
            };

        var attributeDefinitions = new List<AttributeDefinition>()
    {
        // Attribute definitions for table primary key
        { new AttributeDefinition() {
              AttributeName = "CustomerId", AttributeType = "S"
          } },
        { new AttributeDefinition() {
              AttributeName = "OrderId", AttributeType = "N"
          } },
        // Attribute definitions for index primary key
        { new AttributeDefinition() {
              AttributeName = "OrderCreationDate", AttributeType = "N"
          } },
        { new AttributeDefinition() {
```

```
                AttributeName = "IsOpen", AttributeType = "N"
            }}
        };

            createTableRequest.AttributeDefinitions = attributeDefinitions;

            // Key schema for table
            var tableKeySchema = new List<KeySchemaElement>()
        {
            { new KeySchemaElement() {
                AttributeName = "CustomerId", KeyType = "HASH"
            } },                                                    //Partition key
            { new KeySchemaElement() {
                AttributeName = "OrderId", KeyType = "RANGE"
            } }                                                   //Sort key
        };

            createTableRequest.KeySchema = tableKeySchema;

            var localSecondaryIndexes = new List<LocalSecondaryIndex>();

            // OrderCreationDateIndex
            LocalSecondaryIndex orderCreationDateIndex = new LocalSecondaryIndex()
            {
                IndexName = "OrderCreationDateIndex"
            };

            // Key schema for OrderCreationDateIndex
            var indexKeySchema = new List<KeySchemaElement>()
        {
            { new KeySchemaElement() {
                AttributeName = "CustomerId", KeyType = "HASH"
            } },                                                      //Partition key
            { new KeySchemaElement() {
                AttributeName = "OrderCreationDate", KeyType = "RANGE"
            } }                                                                 //Sort key
        };

            orderCreationDateIndex.KeySchema = indexKeySchema;

            // Projection (with list of projected attributes) for
            // OrderCreationDateIndex
            var projection = new Projection()
            {
```

```
            ProjectionType = "INCLUDE"
        };

        var nonKeyAttributes = new List<string>()
    {
        "ProductCategory",
        "ProductName"
    };
        projection.NonKeyAttributes = nonKeyAttributes;

        orderCreationDateIndex.Projection = projection;

        localSecondaryIndexes.Add(orderCreationDateIndex);

        // IsOpenIndex
        LocalSecondaryIndex isOpenIndex
            = new LocalSecondaryIndex()
            {
                IndexName = "IsOpenIndex"
            };

        // Key schema for IsOpenIndex
        indexKeySchema = new List<KeySchemaElement>()
    {
        { new KeySchemaElement() {
              AttributeName = "CustomerId", KeyType = "HASH"
          }},                                                    //Partition key
        { new KeySchemaElement() {
              AttributeName = "IsOpen", KeyType = "RANGE"
          }}                                                  //Sort key
    };

        // Projection (all attributes) for IsOpenIndex
        projection = new Projection()
        {
            ProjectionType = "ALL"
        };

        isOpenIndex.KeySchema = indexKeySchema;
        isOpenIndex.Projection = projection;

        localSecondaryIndexes.Add(isOpenIndex);

        // Add index definitions to CreateTable request
```

```
            createTableRequest.LocalSecondaryIndexes = localSecondaryIndexes;

            Console.WriteLine("Creating table " + tableName + "...");
            client.CreateTable(createTableRequest);
            WaitUntilTableReady(tableName);
        }

        public static void Query(string indexName)
        {

Console.WriteLine("\n*******************************************************\n");
            Console.WriteLine("Querying table " + tableName + "...");

            QueryRequest queryRequest = new QueryRequest()
            {
                TableName = tableName,
                ConsistentRead = true,
                ScanIndexForward = true,
                ReturnConsumedCapacity = "TOTAL"
            };


            String keyConditionExpression = "CustomerId = :v_customerId";
            Dictionary<string, AttributeValue> expressionAttributeValues = new
Dictionary<string, AttributeValue> {
            {":v_customerId", new AttributeValue {
                 S = "bob@example.com"
             }}
        };


            if (indexName == "IsOpenIndex")
            {
                Console.WriteLine("\nUsing index: '" + indexName
                          + "': Bob's orders that are open.");
                Console.WriteLine("Only a user-specified list of attributes are
returned\n");
                queryRequest.IndexName = indexName;

                keyConditionExpression += " and IsOpen = :v_isOpen";
                expressionAttributeValues.Add(":v_isOpen", new AttributeValue
                {
                    N = "1"
                });
```

```
                // ProjectionExpression
                queryRequest.ProjectionExpression = "OrderCreationDate,
ProductCategory, ProductName, OrderStatus";
            }
            else if (indexName == "OrderCreationDateIndex")
            {
                Console.WriteLine("\nUsing index: '" + indexName
                          + "': Bob's orders that were placed after 01/31/2013.");
                Console.WriteLine("Only the projected attributes are returned\n");
                queryRequest.IndexName = indexName;

                keyConditionExpression += " and OrderCreationDate > :v_Date";
                expressionAttributeValues.Add(":v_Date", new AttributeValue
                {
                    N = "20130131"
                });

                // Select
                queryRequest.Select = "ALL_PROJECTED_ATTRIBUTES";
            }
            else
            {
                Console.WriteLine("\nNo index: All of Bob's orders, by OrderId:\n");
            }
            queryRequest.KeyConditionExpression = keyConditionExpression;
            queryRequest.ExpressionAttributeValues = expressionAttributeValues;

            var result = client.Query(queryRequest);
            var items = result.Items;
            foreach (var currentItem in items)
            {
                foreach (string attr in currentItem.Keys)
                {
                    if (attr == "OrderId" || attr == "IsOpen"
                        || attr == "OrderCreationDate")
                    {
                        Console.WriteLine(attr + "---> " + currentItem[attr].N);
                    }
                    else
                    {
                        Console.WriteLine(attr + "---> " + currentItem[attr].S);
                    }
                }
```

```
                Console.WriteLine();
            }
            Console.WriteLine("\nConsumed capacity: " +
result.ConsumedCapacity.CapacityUnits + "\n");
        }

        private static void DeleteTable(string tableName)
        {
            Console.WriteLine("Deleting table " + tableName + "...");
            client.DeleteTable(new DeleteTableRequest()
            {
                TableName = tableName
            });
            WaitForTableToBeDeleted(tableName);
        }

        public static void LoadData()
        {
            Console.WriteLine("Loading data into table " + tableName + "...");

            Dictionary<string, AttributeValue> item = new Dictionary<string,
AttributeValue>();

            item["CustomerId"] = new AttributeValue
            {
                S = "alice@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "1"
            };
            item["IsOpen"] = new AttributeValue
            {
                N = "1"
            };
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130101"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Book"
            };
            item["ProductName"] = new AttributeValue
```

```
            {
                S = "The Great Outdoors"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "PACKING ITEMS"
            };
            /* no ShipmentTrackingId attribute */
            PutItemRequest putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "alice@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "2"
            };
            item["IsOpen"] = new AttributeValue
            {
                N = "1"
            };
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130221"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Bike"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Super Mountain"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "ORDER RECEIVED"
```

```
            };
            /* no ShipmentTrackingId attribute */
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "alice@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "3"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130304"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Music"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "A Quiet Interlude"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "IN TRANSIT"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "176493"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
```

```
            ReturnItemCollectionMetrics = "SIZE"
        };
        client.PutItem(putItemRequest);

        item = new Dictionary<string, AttributeValue>();
        item["CustomerId"] = new AttributeValue
        {
            S = "bob@example.com"
        };
        item["OrderId"] = new AttributeValue
        {
            N = "1"
        };
        /* no IsOpen attribute */
        item["OrderCreationDate"] = new AttributeValue
        {
            N = "20130111"
        };
        item["ProductCategory"] = new AttributeValue
        {
            S = "Movie"
        };
        item["ProductName"] = new AttributeValue
        {
            S = "Calm Before The Storm"
        };
        item["OrderStatus"] = new AttributeValue
        {
            S = "SHIPPING DELAY"
        };
        item["ShipmentTrackingId"] = new AttributeValue
        {
            S = "859323"
        };
        putItemRequest = new PutItemRequest
        {
            TableName = tableName,
            Item = item,
            ReturnItemCollectionMetrics = "SIZE"
        };
        client.PutItem(putItemRequest);

        item = new Dictionary<string, AttributeValue>();
        item["CustomerId"] = new AttributeValue
```

```
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "2"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130124"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Music"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "E-Z Listening"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "DELIVERED"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "756943"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "3"
```

```
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130221"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Music"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Symphony 9"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "DELIVERED"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "645193"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "4"
            };
            item["IsOpen"] = new AttributeValue
            {
                N = "1"
            };
            item["OrderCreationDate"] = new AttributeValue
```

```
            {
                N = "20130222"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Hardware"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "Extra Heavy Hammer"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "PACKING ITEMS"
            };
            /* no ShipmentTrackingId attribute */
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "5"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130309"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Book"
            };
            item["ProductName"] = new AttributeValue
            {
```

```
            S = "How To Cook"
        };
        item["OrderStatus"] = new AttributeValue
        {
            S = "IN TRANSIT"
        };
        item["ShipmentTrackingId"] = new AttributeValue
        {
            S = "440185"
        };
        putItemRequest = new PutItemRequest
        {
            TableName = tableName,
            Item = item,
            ReturnItemCollectionMetrics = "SIZE"
        };
        client.PutItem(putItemRequest);

        item = new Dictionary<string, AttributeValue>();
        item["CustomerId"] = new AttributeValue
        {
            S = "bob@example.com"
        };
        item["OrderId"] = new AttributeValue
        {
            N = "6"
        };
        /* no IsOpen attribute */
        item["OrderCreationDate"] = new AttributeValue
        {
            N = "20130318"
        };
        item["ProductCategory"] = new AttributeValue
        {
            S = "Luggage"
        };
        item["ProductName"] = new AttributeValue
        {
            S = "Really Big Suitcase"
        };
        item["OrderStatus"] = new AttributeValue
        {
            S = "DELIVERED"
        };
```

```
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "893927"
            };
            putItemRequest = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
                ReturnItemCollectionMetrics = "SIZE"
            };
            client.PutItem(putItemRequest);

            item = new Dictionary<string, AttributeValue>();
            item["CustomerId"] = new AttributeValue
            {
                S = "bob@example.com"
            };
            item["OrderId"] = new AttributeValue
            {
                N = "7"
            };
            /* no IsOpen attribute */
            item["OrderCreationDate"] = new AttributeValue
            {
                N = "20130324"
            };
            item["ProductCategory"] = new AttributeValue
            {
                S = "Golf"
            };
            item["ProductName"] = new AttributeValue
            {
                S = "PGA Pro II"
            };
            item["OrderStatus"] = new AttributeValue
            {
                S = "OUT FOR DELIVERY"
            };
            item["ShipmentTrackingId"] = new AttributeValue
            {
                S = "383283"
            };
            putItemRequest = new PutItemRequest
            {
```

```
            TableName = tableName,
            Item = item,
            ReturnItemCollectionMetrics = "SIZE"
        };
        client.PutItem(putItemRequest);
    }

    private static void WaitUntilTableReady(string tableName)
    {
        string status = null;
        // Let us wait until table is created. Call DescribeTable.
        do
        {
            System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
            try
            {
                var res = client.DescribeTable(new DescribeTableRequest
                {
                    TableName = tableName
                });

                Console.WriteLine("Table name: {0}, status: {1}",
                          res.Table.TableName,
                          res.Table.TableStatus);
                status = res.Table.TableStatus;
            }
            catch (ResourceNotFoundException)
            {
                // DescribeTable is eventually consistent. So you might
                // get resource not found. So we handle the potential exception.
            }
        } while (status != "ACTIVE");
    }

    private static void WaitForTableToBeDeleted(string tableName)
    {
        bool tablePresent = true;

        while (tablePresent)
        {
            System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
            try
            {
                var res = client.DescribeTable(new DescribeTableRequest
```

```
                    {
                        TableName = tableName
                    });

                    Console.WriteLine("Table name: {0}, status: {1}",
                              res.Table.TableName,
                              res.Table.TableStatus);
                }
                catch (ResourceNotFoundException)
                {
                    tablePresent = false;
                }
            }
        }
    }
}
```

Uso de índices secundarios locales en la AWS CLI de DynamoDB

Puede usar la AWS CLI para crear una tabla de Amazon DynamoDB con uno o más índices secundarios locales, describir los índices de la tabla y utilizarlos para realizar consultas.

Temas

- Creación de una tabla con un índice secundario local
- Descripción de una tabla con un índice secundario local
- Consulta de un índice secundario local

Creación de una tabla con un índice secundario local

Los índices secundarios locales se deben crear al mismo tiempo que la tabla. Para ello, use el parámetro `create-table` y proporcione las especificaciones para uno o más índices secundarios locales. En el ejemplo siguiente, se crea una tabla (`Music`) para contener información sobre las canciones de una colección de música. La clave de partición es `Artist` y la de ordenación, `SongTitle`. Un índice secundario `AlbumTitleIndex` en el atributo `AlbumTitle` facilita las consultas por título de álbum.

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
```

```
        AttributeName=AlbumTitle,AttributeType=S  \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[{\"IndexName\": \"AlbumTitleIndex\",
        \"KeySchema\":[{\"AttributeName\":\"Artist\",\"KeyType\":\"HASH\"},
                       {\"AttributeName\":\"AlbumTitle\",\"KeyType\":\"RANGE\"}],
        \"Projection\":{\"ProjectionType\":\"INCLUDE\",  \"NonKeyAttributes\":[\"Genre
\", \"Year\"]}}]"
```

Debe esperar hasta que DynamoDB cree la tabla y establezca el estado de esta última en `ACTIVE`. A partir de ese momento, puede comenzar a incluir elementos de datos en la tabla. Puede utilizar describe-table Para determinar el estado de la creación de tablas.

Descripción de una tabla con un índice secundario local

Para obtener información acerca de los índices secundarios locales en una tabla, utilice el parámetro `describe-table`. Para cada índice, puede obtener acceso a su nombre, esquema de claves y atributos proyectados.

```
aws dynamodb describe-table --table-name Music
```

Consulta de un índice secundario local

Puede usar la operación `query` en un índice secundario local de un modo bastante parecido a como `query` se usa en una tabla. Debe especificar el nombre del índice, los criterios de consulta de la clave de ordenación del índice y los atributos que desea devolver. En este ejemplo, el índice es `AlbumTitleIndex` y la clave de ordenación del índice es `AlbumTitle`.

Los únicos atributos devueltos son aquellos que se han proyectado en el índice. Puede modificar esta consulta de modo que también seleccione atributos sin clave, pero esto requeriría realizar actividad de recuperación en la tabla, lo que resulta relativamente costoso. Para obtener más información sobre recuperaciones de tablas, consulte Proyecciones de atributos.

```
aws dynamodb query \
    --table-name Music \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v_artist and AlbumTitle = :v_title" \
```

```
    --expression-attribute-values  '{":v_artist":{"S":"Acme Band"},":v_title":
{"S":"Songs About Life"} }'
```

Administración de flujos de trabajo complejos con transacciones de DynamoDB

Amazon DynamoDB Transactions simplifica la experiencia de los desarrolladores que realizan cambios coordinados de "todo o nada" en varios elementos y distintas tablas. Las transacciones proporcionan atomicidad, consistencia, aislamiento y durabilidad (ACID, por sus siglas en inglés) en DynamoDB, lo que le ayuda a mantener la exactitud de los datos en sus aplicaciones.

Puede usar las API de lectura y escritura transaccional de DynamoDB para administrar flujos de trabajo empresarial complejos que requieren agregar, actualizar o eliminar varios elementos en una única operación de tipo "todo o nada". Por ejemplo, un desarrollador de videojuegos puede garantizar que los perfiles de los jugadores se actualicen correctamente cuando intercambien elementos en un juego o realicen compras dentro de este.

Con la API de escritura de transacciones, puede agrupar varias acciones `Put`, `Update`, `Delete` y `ConditionCheck`. A continuación, puede presentarlas como una sola operación `TransactWriteItems` que se realiza correctamente o da error como unidad. Lo mismo sucede con varias acciones `Get`, que se pueden agrupar y enviar como una sola operación `TransactGetItems`.

Habilitar las transacciones para las tablas de DynamoDB no conlleva ningún coste adicional. Usted paga sólo por las lecturas o escrituras que forman parte de su transacción. DynamoDB lleva a cabo dos lecturas o escrituras subyacentes de cada elemento de la transacción: una para preparar la transacción y otra para confirmarla. Estas dos operaciones de lectura/escritura subyacentes están visibles en las métricas de Amazon CloudWatch.

Para comenzar a trabajar con transacciones de DynamoDB, descargue el SDK de AWS o la AWS Command Line Interface (AWS CLI) más recientes. A continuación, siga el Ejemplo de transacciones en DynamoDB.

En las secciones siguientes, se proporciona información general detallada sobre las API de transacciones y cómo usarlas en DynamoDB.

Temas

- Funcionamiento de las Transacciones de Amazon DynamoDB

- Uso de IAM con las transacciones de DynamoDB
- Ejemplo de transacciones en DynamoDB

Funcionamiento de las Transacciones de Amazon DynamoDB

Con Amazon DynamoDB Transactions, puede agrupar varias acciones y enviarlas como una sola operación `TransactWriteItems` o `TransactGetItems` de tipo "todo o nada". En las secciones siguientes, se describen las operaciones de la API, la administración de la capacidad, las prácticas recomendadas y otros detalles sobre el uso de operaciones transaccionales en DynamoDB.

Temas

- API TransactWriteItems
- API TransactGetItems
- Niveles de aislamiento para las transacciones de DynamoDB
- Gestión de conflictos de transacciones en DynamoDB
- Uso de las API transaccionales en DynamoDB Accelerator (DAX)
- Administración de la capacidad para las transacciones
- Prácticas recomendadas para las transacciones
- Uso de las API transaccionales con tablas globales
- Transacciones de DynamoDB en comparación con la biblioteca de transacciones del cliente de AWSLabs

API TransactWriteItems

`TransactWriteItems` es una operación de escritura síncrona e idempotente que agrupa hasta 100 acciones de escritura en una única operación de tipo “todo o nada”. Estas acciones pueden estar dirigidas a un máximo de 100 elementos diferenciados en una o varias tablas de DynamoDB en la misma cuenta de AWS y en la misma región. El tamaño total de los elementos de la transacción no puede superar 4 MB. Las acciones se realizan atómicamente, de tal forma que se llevan a cabo correctamente todas o ninguna de ellas.

> **Note**
>
> - Una operación `TransactWriteItems` se diferencia de una operación `BatchWriteItem` en que todas las acciones que contiene deben completarse correctamente o, de lo

contrario, no se realiza ningún cambio. Con una operación `BatchWriteItem`, es posible que solo algunas de las acciones del lote se realicen correctamente y otras, no.

- Las transacciones no se pueden realizar mediante índices.

No se pueden dirigir varias operaciones de la misma transacción al mismo elemento. Por ejemplo, no se puede realizar una acción `ConditionCheck` y también una acción `Update` en el mismo elemento de la misma transacción.

Puede agregar los tipos de acciones siguientes a una transacción:

- `Put`: Inicia una operación `PutItem` para crear un nuevo elemento o sustituir uno antiguo por otro nuevo, ya sea de forma condicional o sin especificar ninguna condición.
- `Update`: Inicia una operación `UpdateItem` para editar los atributos de un elemento existente o agregar un nuevo elemento a la tabla, si no existe ya. Esta acción se utiliza para agregar, eliminar o actualizar atributos a un elemento existente, de forma condicional o sin condiciones.
- `Delete`: Inicia una operación `DeleteItem` para eliminar un solo elemento de una tabla identificado por su clave principal.
- `ConditionCheck`: Comprueba la existencia de un elemento o la condición de atributos concretos del elemento.

Cuando se completa una transacción en DynamoDB, los cambios comienzan a propagarse a los índices secundarios globales (GSI), los flujos y las copias de seguridad. Esta propagación se produce gradualmente: los registros de flujo de la misma transacción pueden aparecer en momentos diferentes y pueden intercalarse con registros de otras transacciones. Los consumidores de flujos no deben asumir la atomicidad de la transacción ni las garantías de orden.

Para garantizar una instantánea atómica de los elementos modificados en una transacción, utilice la operación TransactGetItems para leer todos los elementos relevantes juntos. Esta operación proporciona una visión coherente de los datos, lo que garantiza que vea todos los cambios de una transacción completada o ninguno en absoluto.

Debido a que la propagación no es inmediata, si una tabla se restaura desde una copia de seguridad (RestoreTableFromBackup) o se exporta a un momento dado (ExportTableToPointInTime) a mitad de la propagación, puede contener solo algunos de los cambios realizados durante una transacción reciente.

Idempotencia

Opcionalmente, puede incluir un token de cliente al realizar una llamada a `TransactWriteItems` para asegurarse de que la solicitud sea idempotente. Hacer que las transacciones sean idempotentes ayuda a evitar errores de aplicaciones si se envía la misma operación varias veces debido a una interrupción de la conexión u otro problema de conectividad.

Si la llamada a `TransactWriteItems` original se realizó correctamente, las llamadas siguientes a `TransactWriteItems` con el mismo token de cliente también se devolverán correctamente sin hacer ningún cambio. Si se ha establecido el parámetro `ReturnConsumedCapacity`, la llamada `TransactWriteItems` inicial devolverá el número de unidades de capacidad de escritura consumidas al realizar los cambios. Las llamadas siguientes a `TransactWriteItems` con el mismo token de cliente devolverán el número de unidades de capacidad de lectura consumidas al leer el elemento.

Información importante sobre la idempotencia

- Un token de cliente es válido durante 10 minutos desde que finaliza la solicitud que lo utiliza. Transcurridos esos 10 minutos, cualquier solicitud que use el mismo token de cliente se tratará como si fuera nueva. No debe reutilizar el mismo token de cliente para la misma solicitud si han pasado 10 minutos.
- Si repite una solicitud con el mismo token de cliente dentro del periodo de idempotencia de 10 minutos, pero cambia algún otro parámetro de la solicitud, DynamoDB devolverá una excepción `IdempotentParameterMismatch`.

Control de errores de escritura

Las transacciones de escritura no se realizarán correctamente en las siguientes circunstancias:

- Cuando no se cumple alguna de las condiciones de las expresiones de condición.
- Cuando se produce un error de validación de transacción porque hay más de una acción dirigida al mismo elemento en una sola operación `TransactWriteItems`.
- Cuando una solicitud `TransactWriteItems` está en conflicto con una operación `TransactWriteItems` en curso para uno o varios elementos de la solicitud `TransactWriteItems`. En este caso, la solicitud genera un error `TransactionCanceledException`.
- Cuando no hay suficiente capacidad aprovisionada para completar la transacción.

- Cuando el tamaño de un elemento es excesivo (más de 400 KB), un índice secundario local (LSI) se vuelve demasiado grande o se produce algún error de validación semejante a causa de los cambios realizados por la transacción.
- Cuando hay algún error de usuario, como un formato de datos no válido.

Para obtener más información acerca de cómo se manejan los conflictos con operaciones `TransactWriteItems`, consulte Gestión de conflictos de transacciones en DynamoDB.

API TransactGetItems

`TransactGetItems` es una operación de lectura síncrona que agrupa hasta 100 acciones `Get`. Estas acciones pueden estar dirigidas a un máximo de 100 elementos diferenciados en una o varias tablas de DynamoDB en la misma cuenta y región de AWS. El tamaño total de los elementos de la transacción no puede superar 4 MB.

Las acciones `Get` se realizan atómicamente, de tal forma que se llevan a cabo correctamente todas o ninguna de ellas:

- `Get`: Inicia una operación `GetItem` para recuperar un conjunto de atributos para el elemento que tiene la clave principal especificada. Si no se encuentra ningún elemento que coincida, `Get` no devuelve ningún dato.

Control de errores de lectura

Las transacciones de lectura no se realizarán correctamente en las siguientes circunstancias:

- Cuando una solicitud `TransactGetItems` está en conflicto con una operación `TransactWriteItems` en curso para uno o varios elementos de la solicitud `TransactGetItems`. En este caso, la solicitud genera un error `TransactionCanceledException`.
- Cuando no hay suficiente capacidad aprovisionada para completar la transacción.
- Cuando hay algún error de usuario, como un formato de datos no válido.

Para obtener más información acerca de cómo se manejan los conflictos con operaciones `TransactGetItems`, consulte Gestión de conflictos de transacciones en DynamoDB.

Niveles de aislamiento para las transacciones de DynamoDB

Los niveles de aislamiento de las operaciones transaccionales (`TransactWriteItems` o `TransactGetItems`) y otras operaciones son los siguientes.

SERIALIZABLE

El aislamiento serializable garantiza que los resultados de varias operaciones concurrentes sean iguales que si ninguna operación se hubiera iniciado antes de finalizar la anterior.

Existe aislamiento serializable entre los siguientes tipos de operaciones:

- Entre cualquier operación transaccional y cualquier operación de escritura estándar (`PutItem`, `UpdateItem` o `DeleteItem`).
- Entre cualquier operación transaccional y cualquier operación de lectura estándar (`GetItem`).
- Entre una operación `TransactWriteItems` y una operación `TransactGetItems`.

Aunque existe aislamiento serializable entre las operaciones transaccionales y cada escritura estándar individual escribe en una operación `BatchWriteItem`, no se produce aislamiento serializable entre la transacción y la operación `BatchWriteItem` como una unidad.

Del mismo modo, el nivel de aislamiento entre una operación transaccional y una operación `GetItems` individual de una operación `BatchGetItem` es serializable. Pero el nivel de aislamiento entre la transacción y la operación `BatchGetItem` como unidad es de lectura confirmada.

Una sola solicitud `GetItem` es serializable con respecto a una solicitud `TransactWriteItems` de una de las dos formas siguientes: antes o después de la solicitud `TransactWriteItems`. Multiple solicitudes `GetItem`, de las claves en solicitudes `TransactWriteItems` se pueden ejecutar en cualquier orden, y por lo tanto los resultados son lectura confirmada.

Por ejemplo, si las solicitudes `GetItem` para el elemento A y el elemento B se ejecutan simultáneamente con una solicitud `TransactWriteItems` que modifica tanto el punto A como el punto B, hay cuatro posibilidades:

- Ambos `GetItem` se ejecutan antes de que la solicitud `TransactWriteItems`.
- Ambos `GetItem` se ejecutan después de la solicitud `TransactWriteItems`.
- Solicitud `GetItem` para el elemento A se ejecuta antes de la solicitud `TransactWriteItems`. Para el elemento B, `GetItem` se ejecuta después de `TransactWriteItems`.

- Solicitud `GetItem` para el elemento B se ejecuta antes de la solicitud `TransactWriteItems`. Para el elemento A, `GetItem` se ejecuta después de `TransactWriteItems`.

Utilice `TransactGetItems`, si prefiere el nivel de aislamiento serializable para múltiples solicitudes `GetItem`.

Si se realiza una lectura no transaccional de varios elementos que formaban parte de la misma solicitud de escritura de transacción durante el vuelo, es posible que pueda leer el nuevo estado de algunos de los elementos y el estado anterior de los demás. Solo podrá leer el nuevo estado de todos los elementos que formaban parte de la solicitud de escritura de transacción cuando reciba una respuesta correcta a la escritura de la transacción, lo que indica que la transacción se ha completado.

Una vez que la transacción se completa correctamente y se recibe una respuesta, las operaciones de lectura coherente posterior pueden seguir devolviendo el estado anterior durante un breve periodo debido al modelo de consistencia posterior de DynamoDB. Para garantizar la lectura de los datos más actualizados inmediatamente después de una transacción, debe utilizar lecturas altamente coherentes mediante el establecimiento de `ConsistentRead` a true.

LECTURA CONFIRMADA

El aislamiento de lectura confirmada garantiza que las operaciones de lectura siempre devuelven valores confirmados para un elemento: la lectura nunca presentará una vista al elemento que represente un estado de una escritura transaccional que no haya tenido éxito finalmente. El aislamiento de lectura confirmada no impide las modificaciones del elemento inmediatamente después de la operación de lectura.

El nivel de aislamiento es de lectura confirmada entre cualquier operación transaccional y cualquier operación de lectura que conlleve varias lecturas estándar (`BatchGetItem`, `Query` o `Scan`). Si una escritura transaccional actualiza un elemento en medio de una operación de `BatchGetItem`, `Query` o `Scan`, la parte posterior de la operación de lectura devuelve el nuevo valor confirmado (con `ConsistentRead`) o posiblemente un valor confirmado anterior [lecturas coherentes posteriores]).

Resumen de la operación

A modo de resumen, en la siguiente tabla se indican los niveles de aislamiento entre una operación transaccional (`TransactWriteItems` o `TransactGetItems`) y otras operaciones.

Operación	Nivel de aislamiento
`DeleteItem`	Serializable
`PutItem`	Serializable
`UpdateItem`	Serializable
`GetItem`	Serializable
`BatchGetItem`	Lectura confirmada*
`BatchWriteItem`	NO serializable*
`Query`	Lectura confirmada
`Scan`	Lectura confirmada
Otra operación transaccional	Serializable

Los niveles marcados con un asterisco (*) se aplican a la operación como una unidad. Sin embargo, las acciones individuales contenidas en esas operaciones tienen el nivel de aislamiento serializable.

Gestión de conflictos de transacciones en DynamoDB

Un conflicto de transacciones puede producirse durante solicitudes concurrentes de nivel de elemento en un elemento dentro de una transacción. Los conflictos de transacciones pueden producirse en los siguientes escenarios:

- Una solicitud `PutItem`, `UpdateItem` o `DeleteItem` de un elemento entra en conflicto con una solicitud `TransactWriteItems` en curso que incluye el mismo elemento.
- Un elemento incluido en una solicitud `TransactWriteItems` forma parte de otra solicitud `TransactWriteItems` en curso.
- Un elemento incluido en una solicitud `TransactGetItems` forma parte de otra solicitud `TransactWriteItems`, `BatchWriteItem`, `PutItem`, `UpdateItem` o `DeleteItem` en curso.

> **Note**
>
> - Cuando se rechaza una solicitud `PutItem`, `UpdateItem` o `DeleteItem`, la solicitud no se realiza correctamente y genera una excepción `TransactionConflictException`.
> - Si se rechaza cualquier solicitud de nivel de elemento dentro de una operación `TransactWriteItems` o `TransactGetItems`, la solicitud no se realiza correctamente y genera una excepción `TransactionCanceledException`. Si esa solicitud falla, los SDK de AWS no vuelven a intentar la solicitud.
>
> Si se está utilizando el AWS SDK para Java, la excepción contiene la lista de CancellationReasons, ordenada según la lista de elementos del parámetro `TransactItems` de la solicitud. Para los demás lenguajes, se incluye una representación de cadena de la lista en el mensaje de error de la excepción.
> - Si una operación `TransactWriteItems` o `TransactGetItems` está en conflicto con una solicitud `GetItem` concurrente, las dos operaciones podrían realizarse correctamente.

La métrica TransactionConflict de CloudWatch se incrementa para cada solicitud de nivel de elemento que no se realiza correctamente.

Uso de las API transaccionales en DynamoDB Accelerator (DAX)

Tanto `TransactWriteItems` como `TransactGetItems` se admiten en DynamoDB Accelerator (DAX) con los mismos niveles de aislamiento que en DynamoDB.

`TransactWriteItems` escribe a través de DAX. DAX pasa una llamada `TransactWriteItems` a DynamoDB y devuelve la respuesta. Para rellenar la caché después de la escritura, DAX llama a `TransactGetItems` en segundo plano para cada elemento en la operación `TransactWriteItems`, que consume unidades de capacidad de lectura adicionales. (Para obtener más información, consulte). Administración de la capacidad para las transacciones.) Esta funcionalidad le permite no complicar la lógica de la aplicación y utilizar DAX para operaciones transaccionales así como para operaciones no transaccionales.

`TransactGetItems`Las llamadas se pasan por DAX sin que los elementos se almacenen en caché localmente. Es el mismo comportamiento que tienen las API de lectura de consistencia alta en DAX.

Administración de la capacidad para las transacciones

Habilitar las transacciones para las tablas de DynamoDB no conlleva ningún coste adicional. Usted paga sólo por las lecturas o escrituras que forman parte de su transacción. DynamoDB lleva a cabo dos lecturas o escrituras subyacentes de cada elemento de la transacción: una para preparar la transacción y otra para confirmarla. Las dos operaciones de lectura/escritura subyacentes están visibles en las métricas de Amazon CloudWatch.

Planee las lecturas y escrituras adicionales que requieren las API transaccionales al aprovisionar la capacidad para las tablas. Por ejemplo, supongamos que su aplicación ejecuta una transacción por segundo y que cada transacción escribe tres elementos de 500 bites en la tabla. Cada elemento requiere dos unidades de capacidad de escritura (WCU): una para preparar la transacción y otra para confirmarla. Por consiguiente, tendría que aprovisionar seis WCU para la tabla.

Si utilizó DynamoDB Accelerator (DAX) en el ejemplo anterior, también debió utilizar dos unidades de capacidad de lectura (RCU) para cada elemento de la llamada `TransactWriteItems`. Por tanto, tendría que aprovisionar seis RCU adicionales para la tabla.

Del mismo modo, si la aplicación ejecuta una transacción de lectura por segundo y cada transacción lee tres elementos de 500 bytes de la tabla, debería aprovisionar seis unidades de capacidad de lectura (RCU) para la tabla. La lectura de cada elemento requiere dos RCU: una para preparar la transacción y otra para confirmarla.

Además, el comportamiento predeterminado del SDK consiste en reintentar las transacciones si se produce alguna excepción `TransactionInProgressException`. Planee las unidades de capacidad de lectura (RCU) adicionales que consumen estos reintentos. Lo mismo sucede si reintenta transacciones de su propio código mediante un `ClientRequestToken`.

Prácticas recomendadas para las transacciones

Es importante tener en cuenta las prácticas recomendadas siguientes al utilizar transacciones de DynamoDB.

- Habilite el escalado automático en las tablas o asegúrese de haber aprovisionado suficiente capacidad de rendimiento para llevar a cabo las dos operaciones de lectura o escritura para cada elemento de la transacción.
- Si no utiliza un SDK proporcionado por AWS, incluya un atributo `ClientRequestToken` al realizar una llamada a `TransactWriteItems` para asegurarse de que la solicitud sea idempotente.

- No agrupe las operaciones en una transacción si no es necesario. Por ejemplo, si una misma transacción compuesta de 10 operaciones se puede desglosar en varias transacciones de modo que la aplicación continúe siendo correcta, recomendamos dividir la transacción. Usar transacciones más sencillas mejora el rendimiento y aumenta la probabilidad de que se ejecuten correctamente.
- Si varias transacciones actualizan los mismos elementos de forma simultánea, pueden provocar conflictos que cancelen las transacciones. Recomendamos las siguientes prácticas recomendadas de DynamoDB para modelar los datos de tal forma que se reduzcan al mínimo esos conflictos.
- Si un conjunto de atributos se actualiza a menudo para varios elementos durante una misma transacción, puede ser conveniente agrupar los atributos en un solo elemento para reducir el ámbito de la transacción.
- Evite utilizar transacciones para la ingesta masiva de datos. Para las escrituras masivas, es preferible usar `BatchWriteItem`.

Uso de las API transaccionales con tablas globales

Las operaciones transaccionales proporcionan garantías de atomicidad, uniformidad, aislamiento y durabilidad (ACID) solo en la región de AWS en la que se invocó la API de escritura. No se admiten las transacciones entre regiones en las tablas globales. Por ejemplo, supongamos que tiene una tabla global con réplicas en las regiones EE. UU. Este (Ohio) y EE. UU. Oeste (Oregón) y realiza una operación `TransactWriteItems` en la región EE. UU. Este (Norte de Virginia). Puede observar transacciones parcialmente completadas en la región Oeste de EE. UU. (Oregón) a medida que se replican los cambios. Los cambios se replican en otras regiones solo cuando se han confirmado en la región de origen.

Transacciones de DynamoDB en comparación con la biblioteca de transacciones del cliente de AWSLabs

Las transacciones de DynamoDB proporcionan una forma más rentable, robusta y eficiente de sustituir la biblioteca de cliente de transacciones de AWSLabs. Sugerimos actualizar las aplicaciones de modo que utilicen las API de transacciones nativas del lado del servidor.

Uso de IAM con las transacciones de DynamoDB

Puede usar AWS Identity and Access Management (IAM) para restringir las acciones que las operaciones transaccionales pueden ejecutar en Amazon DynamoDB. Para obtener más información

sobre el uso de políticas de IAM en DynamoDB, consulte Políticas basadas en identidad de DynamoDB.

Los permisos para las acciones `Put`, `Update`, `Delete` y `Get` se rigen por los permisos usados para las operaciones `PutItem`, `UpdateItem`, `DeleteItem` y `GetItem` subyacentes. Para la acción `ConditionCheck`, puede usar el permiso `dynamodb:ConditionCheckItem` en las políticas de IAM.

A continuación, se proporcionan ejemplos de políticas de IAM que puede utilizar para configurar las transacciones de DynamoDB.

Ejemplo 1: permitir las operaciones transaccionales

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ]
        }
    ]
}
```

Ejemplo 2: permitir solo las operaciones transaccionales

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "dynamodb:EnclosingOperation": [
                        "TransactWriteItems",
                        "TransactGetItems"
                    ]
                }
            }
        }
    ]
}
```

Ejemplo 3: permitir las lecturas y escrituras no transaccionales y bloquear las lecturas y escrituras transaccionales

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
```

```
                "dynamodb:GetItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:*:table/table04"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "dynamodb:EnclosingOperation": [
                        "TransactWriteItems",
                        "TransactGetItems"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
             "Action": [
                 "dynamodb:PutItem",
                 "dynamodb:DeleteItem",
                 "dynamodb:GetItem",
                 "dynamodb:UpdateItem"
             ],
             "Resource": [
                 "arn:aws:dynamodb:*:*:table/table04"
             ]
         }
    ]
}
```

Ejemplo 4: evitar que se devuelva información en caso de error de comprobación de condición

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ConditionCheckItem",
```

```
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/table01",
            "Condition": {
                "StringEqualsIfExists": {
                    "dynamodb:ReturnValues": "NONE"
                }
            }
        }
    ]
}
```

Ejemplo de transacciones en DynamoDB

Como ejemplo de una situación en la que Amazon DynamoDB Transactions pueden ser útiles, considere esta aplicación Java de ejemplo para un mercado en línea.

La aplicación tiene tres tablas de DynamoDB en el backend:

- `Customers`: en esta tabla se almacenan detalles sobre los clientes del mercado. Su clave principal es un identificador único `CustomerId`.
- `ProductCatalog`: en esta tabla se almacenan detalles tales como el precio y la disponibilidad de los productos que se venden en el mercado. Su clave principal es un identificador único `ProductId`.
- `Orders`: en esta tabla se almacenan detalles sobre los pedidos del mercado. Su clave principal es un identificador único `OrderId`.

Hacer un pedido

Los siguientes fragmentos de código ilustran cómo utilizar las transacciones de DynamoDB para coordinar los múltiples pasos necesarios para crear y procesar un pedido. El uso de una única operación todo o nada garantiza que si falla alguna parte de la transacción, no se ejecuten acciones en la transacción y no se realicen cambios.

En este ejemplo, configura un pedido de un cliente cuyo `customerId` es `09e8e9c8-ec48`. A continuación, se ejecuta como una única transacción mediante el siguiente flujo de trabajo de procesamiento de pedidos:

1. Determine que el ID del cliente sea válido.
2. Asegúrese de que el producto sea `IN_STOCK` y actualice el estado del producto a `SOLD`.
3. Asegúrese de que el pedido aún no existe y cree el pedido.

Validar al cliente

Primero, defina una acción para verificar que un cliente con `customerId` igual a `09e8e9c8-ec48` existe en la tabla de clientes.

```
final String CUSTOMER_TABLE_NAME = "Customers";
final String CUSTOMER_PARTITION_KEY = "CustomerId";
final String customerId = "09e8e9c8-ec48";
final HashMap<String, AttributeValue> customerItemKey = new HashMap<>();
customerItemKey.put(CUSTOMER_PARTITION_KEY, new AttributeValue(customerId));

ConditionCheck checkCustomerValid = new ConditionCheck()
    .withTableName(CUSTOMER_TABLE_NAME)
    .withKey(customerItemKey)
    .withConditionExpression("attribute_exists(" + CUSTOMER_PARTITION_KEY + ")");
```

Actualizar el estado del producto

A continuación, defina una acción para actualizar el estado del producto a `SOLD` si la condición en la que el estado del producto está establecido actualmente en `IN_STOCK` es `true`. Configuración del parámetro `ReturnValuesOnConditionCheckFailure` devuelve el elemento si el atributo de estado del producto del artículo no era igual a `IN_STOCK`.

```
final String PRODUCT_TABLE_NAME = "ProductCatalog";
final String PRODUCT_PARTITION_KEY = "ProductId";
HashMap<String, AttributeValue> productItemKey = new HashMap<>();
productItemKey.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));

Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
expressionAttributeValues.put(":new_status", new AttributeValue("SOLD"));
expressionAttributeValues.put(":expected_status", new AttributeValue("IN_STOCK"));
```

```
Update markItemSold = new Update()
    .withTableName(PRODUCT_TABLE_NAME)
    .withKey(productItemKey)
    .withUpdateExpression("SET ProductStatus = :new_status")
    .withExpressionAttributeValues(expressionAttributeValues)
    .withConditionExpression("ProductStatus = :expected_status")

 .withReturnValuesOnConditionCheckFailure(ReturnValuesOnConditionCheckFailure.ALL_OLD);
```

Crear el pedido

Por último, cree el pedido siempre y cuando un pedido con ese `OrderId` no existe.

```
final String ORDER_PARTITION_KEY = "OrderId";
final String ORDER_TABLE_NAME = "Orders";

HashMap<String, AttributeValue> orderItem = new HashMap<>();
orderItem.put(ORDER_PARTITION_KEY, new AttributeValue(orderId));
orderItem.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));
orderItem.put(CUSTOMER_PARTITION_KEY, new AttributeValue(customerId));
orderItem.put("OrderStatus", new AttributeValue("CONFIRMED"));
orderItem.put("OrderTotal", new AttributeValue("100"));

Put createOrder = new Put()
    .withTableName(ORDER_TABLE_NAME)
    .withItem(orderItem)

 .withReturnValuesOnConditionCheckFailure(ReturnValuesOnConditionCheckFailure.ALL_OLD)
    .withConditionExpression("attribute_not_exists(" + ORDER_PARTITION_KEY + ")");
```

Ejecutar la transacción

En el siguiente ejemplo de la se muestra cómo ejecutar las acciones definidas anteriormente como una única operación de tipo “todo o nada”.

```
    Collection<TransactWriteItem> actions = Arrays.asList(
        new TransactWriteItem().withConditionCheck(checkCustomerValid),
        new TransactWriteItem().withUpdate(markItemSold),
        new TransactWriteItem().withPut(createOrder));

    TransactWriteItemsRequest placeOrderTransaction = new TransactWriteItemsRequest()
        .withTransactItems(actions)
        .withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);
```

```
    // Run the transaction and process the result.
    try {
        client.transactWriteItems(placeOrderTransaction);
        System.out.println("Transaction Successful");

    } catch (ResourceNotFoundException rnf) {
        System.err.println("One of the table involved in the transaction is not found"
 + rnf.getMessage());
    } catch (InternalServerErrorException ise) {
        System.err.println("Internal Server Error" + ise.getMessage());
    } catch (TransactionCanceledException tce) {
        System.out.println("Transaction Canceled " + tce.getMessage());
    }
```

Leer los detalles del pedido

En el siguiente ejemplo se muestra cómo leer el pedido completado transaccionalmente a través de las tablas `Orders` y `ProductCatalog`.

```
HashMap<String, AttributeValue> productItemKey = new HashMap<>();
productItemKey.put(PRODUCT_PARTITION_KEY, new AttributeValue(productKey));

HashMap<String, AttributeValue> orderKey = new HashMap<>();
orderKey.put(ORDER_PARTITION_KEY, new AttributeValue(orderId));

Get readProductSold = new Get()
    .withTableName(PRODUCT_TABLE_NAME)
    .withKey(productItemKey);
Get readCreatedOrder = new Get()
    .withTableName(ORDER_TABLE_NAME)
    .withKey(orderKey);

Collection<TransactGetItem> getActions = Arrays.asList(
    new TransactGetItem().withGet(readProductSold),
    new TransactGetItem().withGet(readCreatedOrder));

TransactGetItemsRequest readCompletedOrder = new TransactGetItemsRequest()
    .withTransactItems(getActions)
    .withReturnConsumedCapacity(ReturnConsumedCapacity.TOTAL);

// Run the transaction and process the result.
try {
```

```
    TransactGetItemsResult result = client.transactGetItems(readCompletedOrder);
    System.out.println(result.getResponses());
} catch (ResourceNotFoundException rnf) {
    System.err.println("One of the table involved in the transaction is not found" +
 rnf.getMessage());
} catch (InternalServerErrorException ise) {
    System.err.println("Internal Server Error" + ise.getMessage());
} catch (TransactionCanceledException tce) {
    System.err.println("Transaction Canceled" + tce.getMessage());
}
```

Captura de datos de cambio con Amazon DynamoDB

Muchas aplicaciones se benefician capturando los cambios en los elementos almacenados en una tabla de DynamoDB en el momento en que se producen. A continuación se muestran algunos ejemplos de casos de uso:

- Una aplicación móvil popular modifica los datos de una tabla de DynamoDB a razón de miles de actualizaciones por segundo. Otra aplicación captura y almacena los datos sobre estas actualizaciones y ofrece métricas de uso de la aplicación para móviles prácticamente en tiempo real.
- Una aplicación financiera modifica los datos del mercado de valores en una tabla de DynamoDB. Diferentes aplicaciones que se ejecutan en paralelo rastrean estos cambios en tiempo real, calculan el valor en riesgo y reequilibran automáticamente las carteras en función de los movimientos del precio de las acciones.
- Los sensores en vehículos de transporte y equipos industriales envían datos a una tabla de DynamoDB. Diferentes aplicaciones monitorean el rendimiento y envían alertas de mensajería cuando se detecta un problema, predicen posibles defectos aplicando algoritmos de aprendizaje automático y comprimen y archivan datos en Amazon Simple Storage Service (Amazon S3).
- Una aplicación envía automáticamente notificaciones a los dispositivos móviles de todos los amigos de un grupo tan pronto como uno de ellos carga una nueva imagen.
- Un nuevo cliente agrega datos a una tabla de DynamoDB. Este evento invoca otra aplicación que envía un mensaje de correo electrónico de bienvenida al nuevo cliente.

DynamoDB admite el streaming de registros de captura de datos de cambio a nivel de elemento en tiempo casi real. Puede crear aplicaciones que consuman estas transmisiones y adopten medidas en función de su contenido.

> **Note**
>
> No se admite agregar etiquetas a DynamoDB Streams ni utilizar el control de acceso basado en atributos (ABAC) con DynamoDB Streams.

El siguiente vídeo le ofrece una introducción al concepto de captación de datos de cambios.

Modos de capacidad de tabla

Temas

- Opciones de streaming para la captura de datos de cambio
- Uso de Kinesis Data Streams para capturar cambios en DynamoDB
- Captura de datos de cambios para DynamoDB Streams

Opciones de streaming para la captura de datos de cambio

DynamoDB ofrece dos modelos de streaming para la captura de datos de cambios: Kinesis Data Streams para DynamoDB y DynamoDB Streams.

Para ayudarle a elegir la solución adecuada para su aplicación, la siguiente tabla resume las características de cada modelo de streaming.

Propiedades	Kinesis Data Streams para DynamoDB	DynamoDB Streams
Retención de datos	Hasta 1 año.	24 horas.
Compatibilidad con Kinesis Client Library (KCL	Admite las versiones 1.X, 2.X y 3.x de KCL.	Admite Versiones 1.X y 2.X de KCL.
Número de consumidores	Hasta 5 consumidores simultáneos por fragmento , o hasta 20 consumidores simultáneos por fragmento con fan-out mejorado.	Hasta 2 consumidores simultáneos por fragmento.

Propiedades	Kinesis Data Streams para DynamoDB	DynamoDB Streams
Cuotas de rendimiento	Sin límite.	Sujeto a las cuotas de rendimiento por tabla de DynamoDB y región de AWS.
Modelo de entrega de registros	Uso de modelos a través de HTTP GetRecords y con fan-out mejorado, Kinesis Data Streams envía los registros a través de HTTP/2 mediante el uso de SubscribeToShard.	Modelo de extracción a través de HTTP utilizando GetRecords.
Ordenación de registros	El atributo timestamp de cada registro de transmisión se puede utilizar para identificar el orden real en el que se produjeron los cambios en la tabla DynamoDB.	Para cada elemento que se modifica de una tabla de DynamoDB, los registros de transmisión aparecen en el mismo orden en que se han realizado las modificaciones del elemento.
Registros duplicados	Los registros duplicados pueden aparecer ocasionalmente en la transmisión.	No aparecen registros duplicados en la transmisión.
Opciones de procesamiento de flujos	Procese los registros de transmisiones mediante AWS Lambda, Amazon Managed Service para Apache Flink, Kinesis data Firehose o ETL de streaming de AWS Glue.	Procesar registros de transmisión mediante AWS Lambda o el Adaptador Kinesis de DynamoDB Streams.
Nivel de durabilidad	Zonas de disponibilidad para proporcionar una conmutación por error automática sin interrupciones.	Zonas de disponibilidad para proporcionar una conmutación por error automática sin interrupciones.

Puede habilitar ambos modelos de streaming en la misma tabla de DynamoDB.

En las siguientes charlas de vídeo, se comparan las diferencias entre las dos opciones.

DynamoDB Streams vs Kinesis Data Streams

Uso de Kinesis Data Streams para capturar cambios en DynamoDB

Puede utilizar Amazon Kinesis Data Streams para capturar cambios en Amazon DynamoDB.

Kinesis Data Streams captura modificaciones a nivel de elemento en cualquier tabla de DynamoDB y las replica en una secuencia de datos de Kinesis. Sus aplicaciones pueden acceder a este flujo de datos y ver los cambios a nivel de elemento casi en tiempo real. Puede capturar y almacenar continuamente terabytes de datos por hora. Puede aprovechar el tiempo de retención de datos más prolongado y, con la capacidad de distribución ramificada mejorada, puede llegar simultáneamente a dos o más aplicaciones descendentes. Otros beneficios incluyen una mayor transparencia de auditoría y seguridad.

Kinesis Data Streams también le da acceso a Amazon Data Firehose y Amazon Managed Service para Apache Flink. Estos servicios pueden ayudarle a crear aplicaciones que alimenten paneles en tiempo real, generen alertas, apliquen precios y publicidad dinámicos y apliquen análisis de datos sofisticados y algoritmos de machine learning.

> **Note**
>
> El uso de Kinesis Data Streams para DynamoDB está sujeto a los precios de Kinesis Data Streams para el flujo de datos y a los precios de DynamoDB para la tabla de origen.

Cómo funciona Kinesis Data Streams con DynamoDB

Cuando un flujo de datos de Kinesis está habilitado para una tabla de DynamoDB, la tabla envía un registro de datos que captura cualquier cambio en los datos de esa tabla. Este registro de datos incluye:

- La hora específica en la que se creó, se actualizó o se eliminó recientemente un elemento
- Clave principal de ese elemento
- Una instantánea del registro antes de la modificación

- Una instantánea del registro después de la modificación

Estos registros de datos se capturan y publican casi en tiempo real. Después de escribirlos en el flujo de datos de Kinesis, se pueden leer igual que cualquier otro registro. Puede utilizar la biblioteca de clientes de Kinesis, utilizar AWS Lambda, llamar a la API de Kinesis Data Streams y utilizar otros servicios conectados. Para obtener más información, consulte Lectura de datos desde Amazon Kinesis Data Streams en la Guía para desarrolladores de Amazon Kinesis Data Streams.

Estos cambios en los datos también se capturan de forma asíncrona. Kinesis no tiene ningún impacto en el rendimiento de una tabla desde la que se transmite. Los registros de transmisión almacenados en la secuencia de datos de Kinesis se cifran en reposo. Para obtener más información, consulte Protección de datos en Amazon Kinesis Data Streams.

Los registros del flujo de datos de Kinesis podrían aparecer en un orden distinto al de cuando se produjeron los cambios en los elementos. Las mismas notificaciones de elementos también podrían aparecer más de una vez en el flujo. Puede verificar el atributo `ApproximateCreationDateTime` para identificar el orden en el que se produjeron las modificaciones de los elementos e identificar los registros duplicados.

Al activar un flujo de datos de Kinesis como destino de transmisión de una tabla de DynamoDB, puede configurar la precisión de los valores `ApproximateCreationDateTime` en milisegundos o microsegundos. De forma predeterminada, `ApproximateCreationDateTime` indica la hora del cambio en milisegundos. Además, puede cambiar este valor en un destino de transmisión activo. Tras dicha actualización, los registros de flujo escritos en Kinesis tendrán valores `ApproximateCreationDateTime` con la precisión deseada.

Los valores binarios escritos en DynamoDB deben estar codificados en formato base64. No obstante, cuando los registros de datos se escriben en un flujo de datos Kinesis, estos valores binarios codificados se codifican en base64 por segunda vez. Al leer estos registros de un flujo de datos Kinesis, para recuperar los valores binarios en bruto, las aplicaciones deben descodificar estos valores dos veces.

DynamoDB cobra por el uso de Kinesis Data Streams en unidades de captura de datos de cambio. 1 KB de cambio por elemento único cuenta como una unidad de captura de datos de cambio. El KB de cambio de cada elemento se calcula por la mayor de las imágenes “antes” y “después” del elemento escrito en el flujo, y se utiliza la misma lógica que el consumo de unidades de capacidad para operaciones de escritura. De manera similar a la que funciona el modo bajo demanda de

DynamoDB, no es necesario aprovisionar el rendimiento de capacidad para las unidades de captura de datos de cambio.

Activación de una instancia de flujo de datos de Kinesis para la tabla de DynamoDB

Puede habilitar o desactivar el streaming a Kinesis en su tabla de DynamoDB existente mediante la Consola de administración de AWS, el SDK de AWS o AWS Command Line Interface (AWS CLI).

- Solo puede transmitir datos de DynamoDB a Kinesis Data Streams en la misma cuenta AWS y región AWS que su tabla.
- Solo puede transmitir datos desde una tabla de DynamoDB a un flujo de datos de Kinesis.

Aplicación de cambios en un destino de Kinesis Data Streams de la tabla de DynamoDB

De forma predeterminada, todos los registros de flujos de datos de Kinesis incluyen un atributo `ApproximateCreationDateTime`. Este atributo representa una marca de tiempo en milisegundos del momento aproximado en que se creó cada registro. Puede cambiar la precisión de estos valores en https://console.aws.amazon.com/kinesis, el SDK o la AWS CLI.

Introducción a Kinesis Data Streams para Amazon DynamoDB

En esta sección se describe cómo usar las tablas de Kinesis Data Streams para Amazon DynamoDB con la consola de Amazon DynamoDB, la AWS Command Line Interface (AWS CLI) y la API.

Creación de un flujo de datos de Amazon Kinesis activo

Todos estos ejemplos utilizan la tabla `Music` de DynamoDB que se creó como parte del tutorial Introducción a DynamoDB.

Para obtener más información sobre cómo crear consumidores y conectar el flujo de datos de Kinesis a otros servicios de AWS, consulte Lectura de datos de Kinesis Data Streams en la Guía para desarrolladores de Amazon Kinesis Data Streams.

> **Note**
>
> Cuando utilice particiones de KDS por primera vez, le recomendamos configurar los fragmentos para escalar verticalmente y reducir verticalmente según los patrones de uso. Cuando haya acumulado más datos sobre los patrones de uso, podrá ajustar las particiones de la transmisión para que coincidan.

Console

1. Inicie sesión en la Consola de administración de AWS y abra la consola de Kinesis en https://console.aws.amazon.com/kinesis/.
2. Seleccionar Creación de flujos de datos y siga las instrucciones para crear una transmisión llamada `samplestream`.
3. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
4. En el panel de navegación del lado izquierdo de la consola, elija Tablas.
5. Elija la tabla Music.
6. Elija la pestaña Exports and streams (Exportaciones y flujos).
7. (Opcional) En Detalles de Amazon Kinesis Data Streams, puede cambiar la precisión de la marca de tiempo del registro de microsegundos (predeterminado) a milisegundos.
8. Elija samplestream de la lista desplegable.
9. Seleccione el botón Activar.

AWS CLI

1. Cree un flujo de datos de Kinesis llamado `samplestream` mediante el uso del comando create-stream.

   ```
   aws kinesis create-stream --stream-name samplestream --shard-count 3
   ```

   Consulte Consideraciones sobre la administración de particiones para Kinesis Data Streams antes de configurar el número de fragmentos para el flujo de datos de Kinesis.
2. Verifique que la transmisión de Kinesis esté activa y lista para su uso mediante el comando describe-stream.

   ```
   aws kinesis describe-stream --stream-name samplestream
   ```
3. Habilite el streaming de Kinesis en la tabla de DynamoDB mediante el `enable-kinesis-streaming-destination` comando. Reemplace el valor de `stream-arn` por el que `describe-stream` devolvió en el paso anterior. Si lo desea, active la transmisión con una precisión más detallada (microsegundos) de los valores de marca de tiempo devueltos en cada registro.

   Active la transmisión con una precisión de marca de tiempo de microsegundos:

```
aws dynamodb enable-kinesis-streaming-destination \
  --table-name Music \
  --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
  --enable-kinesis-streaming-configuration
 ApproximateCreationDateTimePrecision=MICROSECOND
```

O active la transmisión con la precisión de marca de tiempo predeterminada (milisegundos):

```
aws dynamodb enable-kinesis-streaming-destination \
  --table-name Music \
  --stream-arn arn:aws:kinesis:us-west-2:12345678901:stream/samplestream
```

4. Verifique si el streaming de Kinesis está activa en la tabla mediante el comando `describe-kinesis-streaming-destination` de DynamoDB.

```
aws dynamodb describe-kinesis-streaming-destination --table-name Music
```

5. Escribir datos en la tabla de DynamoDB utilizando el comando `put-item`, tal y como se describe en la Guía para desarrolladores de DynamoDB.

```
aws dynamodb put-item \
    --table-name Music  \
    --item \
        '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me
 Today"}, "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N": "1"}}'

aws dynamodb put-item \
    --table-name Music \
    --item \
        '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"},
 "AlbumTitle": {"S": "Songs About Life"}, "Awards": {"N": "10"} }'
```

6. Uso del comando de la CLI get-records para Kinesis para recuperar el contenido del flujo de Kinesis. A continuación, utilice el siguiente fragmento de código para deserializar el contenido de la transmisión.

```
/**
 * Takes as input a Record fetched from Kinesis and does arbitrary processing as
 an example.
 */
```

```
public void processRecord(Record kinesisRecord) throws IOException {
    ByteBuffer kdsRecordByteBuffer = kinesisRecord.getData();
    JsonNode rootNode = OBJECT_MAPPER.readTree(kdsRecordByteBuffer.array());
    JsonNode dynamoDBRecord = rootNode.get("dynamodb");
    JsonNode oldItemImage = dynamoDBRecord.get("OldImage");
    JsonNode newItemImage = dynamoDBRecord.get("NewImage");
    Instant recordTimestamp = fetchTimestamp(dynamoDBRecord);

    /**
     * Say for example our record contains a String attribute named "stringName"
 and we want to fetch the value
     * of this attribute from the new item image. The following code fetches
 this value.
     */
    JsonNode attributeNode = newItemImage.get("stringName");
    JsonNode attributeValueNode = attributeNode.get("S"); // Using DynamoDB "S"
 type attribute
    String attributeValue = attributeValueNode.textValue();
    System.out.println(attributeValue);
}

private Instant fetchTimestamp(JsonNode dynamoDBRecord) {
    JsonNode timestampJson = dynamoDBRecord.get("ApproximateCreationDateTime");
    JsonNode timestampPrecisionJson =
 dynamoDBRecord.get("ApproximateCreationDateTimePrecision");
    if (timestampPrecisionJson != null &&
 timestampPrecisionJson.equals("MICROSECOND")) {
        return Instant.EPOCH.plus(timestampJson.longValue(), ChronoUnit.MICROS);
    }
    return Instant.ofEpochMilli(timestampJson.longValue());
}
```

Java

1. Siga las instrucciones de la guía para desarrolladores de Kinesis Data Streams para crear una secuencia de datos de Kinesis llamada `samplestream` usando Java.

   Consulte Consideraciones sobre la administración de particiones para Kinesis Data Streams antes de configurar el número de fragmentos para el flujo de datos de Kinesis.

2. Use el siguiente fragmento de código para habilitar el streaming de Kinesis en la tabla de DynamoDB. Si lo desea, active la transmisión con una precisión más detallada (microsegundos) de los valores de marca de tiempo devueltos en cada registro.

   Active la transmisión con una precisión de marca de tiempo de microsegundos:

```
EnableKinesisStreamingConfiguration enableKdsConfig =
 EnableKinesisStreamingConfiguration.builder()

 .approximateCreationDateTimePrecision(ApproximateCreationDateTimePrecision.MICROSECOND)
  .build();

EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .enableKinesisStreamingConfiguration(enableKdsConfig)
  .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

   O active la transmisión con la precisión de marca de tiempo predeterminada (milisegundos):

```
EnableKinesisStreamingDestinationRequest enableKdsRequest =
 EnableKinesisStreamingDestinationRequest.builder()
  .tableName(tableName)
  .streamArn(kdsArn)
  .build();

EnableKinesisStreamingDestinationResponse enableKdsResponse =
 ddbClient.enableKinesisStreamingDestination(enableKdsRequest);
```

3. Siga las instrucciones de la Guía para desarrolladores de Kinesis Data Streams para leer desde el flujo de datos creado.

4. Utilice el siguiente fragmento de código para deserializar el contenido del flujo.

```
/**
 * Takes as input a Record fetched from Kinesis and does arbitrary processing as
 an example.
 */
```

```
public void processRecord(Record kinesisRecord) throws IOException {
    ByteBuffer kdsRecordByteBuffer = kinesisRecord.getData();
    JsonNode rootNode = OBJECT_MAPPER.readTree(kdsRecordByteBuffer.array());
    JsonNode dynamoDBRecord = rootNode.get("dynamodb");
    JsonNode oldItemImage = dynamoDBRecord.get("OldImage");
    JsonNode newItemImage = dynamoDBRecord.get("NewImage");
    Instant recordTimestamp = fetchTimestamp(dynamoDBRecord);

    /**
     * Say for example our record contains a String attribute named "stringName"
 and we wanted to fetch the value
     * of this attribute from the new item image, the below code would fetch
 this.
     */
    JsonNode attributeNode = newItemImage.get("stringName");
    JsonNode attributeValueNode = attributeNode.get("S"); // Using DynamoDB "S"
 type attribute
    String attributeValue = attributeValueNode.textValue();
    System.out.println(attributeValue);
}

private Instant fetchTimestamp(JsonNode dynamoDBRecord) {
    JsonNode timestampJson = dynamoDBRecord.get("ApproximateCreationDateTime");
    JsonNode timestampPrecisionJson =
 dynamoDBRecord.get("ApproximateCreationDateTimePrecision");
    if (timestampPrecisionJson != null &&
 timestampPrecisionJson.equals("MICROSECOND")) {
        return Instant.EPOCH.plus(timestampJson.longValue(), ChronoUnit.MICROS);
    }
    return Instant.ofEpochMilli(timestampJson.longValue());
}
```

Aplicación de cambios en un flujo de datos de Amazon Kinesis activo

En esta sección se describe cómo realizar cambios en una configuración de Kinesis Data Streams para DynamoDB desde la consola, la AWS CLI o la API.

Consola de administración de AWS

1. Abra la consola de DynamoDB desde ttps://console.aws.amazon.com/dynamodb/.
2. Busque su tabla.
3. Elija la pestaña Exportaciones y flujos.

AWS CLI

1. Llame a `describe-kinesis-streaming-destination` para confirmar que el flujo es `ACTIVE`.
2. Llame a `UpdateKinesisStreamingDestination`, como en este ejemplo:

```
aws dynamodb update-kinesis-streaming-destination --table-name
 enable_test_table --stream-arn arn:aws:kinesis:us-east-1:12345678901:stream/
enable_test_stream --update-kinesis-streaming-configuration
 ApproximateCreationDateTimePrecision=MICROSECOND
```

3. Llame a `describe-kinesis-streaming-destination` para confirmar que el flujo es `UPDATING`.
4. Llame a `describe-kinesis-streaming-destination` periódicamente hasta que el estado de la transmisión sea `ACTIVE` de nuevo. Las actualizaciones de precisión de la marca de tiempo suelen aplicarse al cabo de unos cinco minutos. Una vez actualizado el estado, significa que la actualización se ha completado y que el nuevo valor de precisión se aplicará a los registros futuros.
5. Escriba en la tabla con `putItem`.
6. Uso el comando `get-records` de Kinesis para recuperar el contenido del flujo.
7. Confirme que el `ApproximateCreationDateTime` de las escrituras tienen la precisión deseada.

API de Java

1. Proporcione un fragmento de código que construya una solicitud `UpdateKinesisStreamingDestination` y una respuesta `UpdateKinesisStreamingDestination`.
2. Proporcione un fragmento de código que construya una solicitud `DescribeKinesisStreamingDestination` y una `DescribeKinesisStreamingDestination response`.
3. Llame a `describe-kinesis-streaming-destination` periódicamente hasta que el estado del flujo sea `ACTIVE` de nuevo, lo que significa que la actualización se ha completado y que el nuevo valor de precisión se aplicará a los registros futuros.
4. Realice escrituras en la tabla.
5. Lea el flujo y deserialice el contenido del flujo.

6. Confirme que el `ApproximateCreationDateTime` de las escrituras tienen la precisión deseada.

Uso de particiones y métricas con DynamoDB Streams y Kinesis Data Streams

Consideraciones sobre la administración de particiones para Kinesis Data Streams

Un flujo de datos de Kinesis cuenta su rendimiento en particiones. En Amazon Kinesis Data Streams, puede elegir entre un modo bajo demanda y un modo aprovisionado para sus flujos de datos.

Se recomienda utilizar el modo bajo demanda para Kinesis Data Streams si la carga de trabajo de escritura de DynamoDB es muy variable e impredecible. Con el modo bajo demanda, no es necesario planificar la capacidad pues Kinesis Data Streams administra automáticamente las particiones para proporcionar el rendimiento necesario.

Para las cargas de trabajo predecibles, puede usar el modo aprovisionado para su flujo de datos de Kinesis. Con el modo aprovisionado, debe especificar el número de particiones del flujo de datos para adaptar los registros de captura de datos de cambios de DynamoDB. Para determinar el número de particiones que necesitará el flujo de datos de Kinesis para admitir la tabla de DynamoDB, necesitará los siguientes valores de entrada:

- El tamaño promedio del registro de su tabla de DynamoDB en bytes (`average_record_size_in_bytes`).
- El número máximo de operaciones de escritura que su tabla de DynamoDB llevará a cabo por segundo. Esto incluye las operaciones de creación, eliminación y actualización realizadas por sus aplicaciones, así como las operaciones generadas automáticamente, como las operaciones de eliminación generadas por Tiempo de vida (`write_throughput`).
- El porcentaje de operaciones de actualización y sobrescritura que lleva a cabo en la tabla en comparación con las operaciones de creación o eliminación (`percentage_of_updates`). Tenga en cuenta que las operaciones de actualización y sobrescritura replican tanto las imágenes antiguas como las nuevas del elemento modificado en el flujo. Esto genera el doble del tamaño del elemento de DynamoDB.

Puede calcular el número de particiones (`number_of_shards`) que necesita su flujo de datos de Kinesis con los valores de entrada de la siguiente fórmula:

```
number_of_shards = ceiling( max( ((write_throughput * (4+percentage_of_updates) *
 average_record_size_in_bytes) / 1024 / 1024), (write_throughput/1000)), 1)
```

Por ejemplo, es posible que tenga un rendimiento máximo de 1040 operaciones de escritura por segundo (`write_throughput`) con un tamaño de registro medio de 800 bytes (`average_record_size_in_bytes`)). Si el 25 % de esas operaciones de escritura son operaciones de actualización (`percentage_of_updates`), necesitará dos particiones (`number_of_shards`) para adaptarse a su rendimiento de transmisión de DynamoDB:

```
ceiling( max( ((1040 * (4+25/100) * 800)/ 1024 / 1024), (1040/1000)), 1).
```

Tenga en cuenta lo siguiente antes de utilizar la fórmula para calcular el número de particiones necesarias con el modo aprovisionado para los flujos de datos de Kinesis:

- Esta fórmula ayuda a realizar una estimación del número de particiones que se necesitará para acomodar los flujos de datos de cambios de DynamoDB. No representa el número total de particiones necesario en el flujo de datos de Kinesis, como el número de particiones necesario para admitir a los consumidores del flujo de datos de Kinesis.
- En el modo aprovisionado, es posible que se produzcan excepciones en el rendimiento de lectura y escritura si no se configura el flujo de datos para manejar los picos de rendimiento. En este caso, debe escalar manualmente su flujo de datos para acomodar el tráfico de datos.
- Esta fórmula tiene en cuenta la sobrecarga adicional que genera DynamoDB antes de transmitir los registros de datos de los registros de cambios al flujo de datos de Kinesis.

Para obtener más información sobre los modos de capacidad de Kinesis Data Streams, consulte Choosing the Data Stream Capacity Mode. Para obtener más información sobre la diferencia de precios entre los distintos modos de capacidad, consulte Precios de Amazon Kinesis Data Streams.

Monitoreo de la captura de datos de cambios con Kinesis Data Streams

DynamoDB proporciona varias métricas de Amazon CloudWatch para ayudarlo a monitorear la replicación de la captura de datos de cambio en Kinesis. Para obtener una lista completa de las métricas de CloudWatch, consulte Dimensiones y métricas de DynamoDB.

Para determinar si el flujo tiene suficiente capacidad, le recomendamos que monitoree los siguientes elementos tanto durante la habilitación del flujo como en la producción:

- `ThrottledPutRecordCount`: número de registros que el flujo de datos de Kinesis ha limitado porque la capacidad del flujo de datos de Kinesis es insuficiente. Es posible que experimente cierta limitación controlada durante picos de uso excepcionales, pero la `ThrottledPutRecordCount` debería ser lo más baja posible. DynamoDB vuelve a intentar enviar registros limitados al flujo de datos de Kinesis, pero esto podría dar lugar a una mayor latencia de replicación.

  Si experimenta una limitación controlada excesiva y regular, es posible que tenga que aumentar el número de fragmentos del flujo de Kinesis proporcionalmente al rendimiento de escritura observado de la tabla. Para obtener más información sobre cómo determinar el tamaño de un flujo de datos de Kinesis, consulte Determinar el tamaño inicial de un flujo de datos de Kinesis.

- `AgeOfOldestUnreplicatedRecord`: el tiempo transcurrido desde el cambio de nivel de elemento más antiguo que aún no se ha replicado en el flujo de datos de Kinesis apareció en la tabla de DynamoDB. En funcionamiento normal, `AgeOfOldestUnreplicatedRecord` debe estar en el orden de milisegundos. Este número crece según los intentos de replicación fallidos cuando se deben a elecciones de configuración controladas por el cliente.

  Si la métrica `AgeOfOldestUnreplicatedRecord` supera 168 horas, la replicación de los cambios en el nivel de elemento de la tabla de DynamoDB al flujo de datos de Kinesis se desactivará automáticamente.

  Los ejemplos de las configuraciones controladas por el cliente que provocan intentos de replicación fallidos son una capacidad de flujo de datos de Kinesis no aprovisionada que produce una limitación excesiva o una actualización manual de las políticas de acceso del flujo de datos de Kinesis que evita que DynamoDB agregue datos al flujo de datos. Para mantener esta métrica lo más baja posible, es posible que tenga que garantizar el aprovisionamiento correcto de la capacidad del flujo de datos de Kinesis y asegurarse de que los permisos de DynamoDB no se modifiquen.

- `FailedToReplicateRecordCount`: número de registros que DynamoDB no pudo replicar en el flujo de datos de Kinesis. Algunos elementos de más de 34 KB podrían expandirse de tamaño para cambiar los registros de datos que superan el límite de tamaño del elemento de 1 MB de Kinesis Data Streams. Esta expansión de tamaño se produce cuando estos elementos mayores a 34 KB incluyen un gran número de valores de atributos booleanos o vacíos. Los valores de atributos booleanos y vacíos se almacenan como 1 byte en DynamoDB, pero se expanden hasta 5 bytes cuando se serializan mediante JSON estándar para la replicación de Kinesis Data Streams. DynamoDB no puede replicar tales registros de cambios en el flujo de datos de Kinesis. DynamoDB omite estos registros de datos de cambios y continúa replicando automáticamente los registros posteriores.

Puede crear alarmas de Amazon CloudWatch que envíen un mensaje de Amazon Simple Notification Service (Amazon SNS) para notificación cuando cualquiera de las métricas anteriores supere un umbral específico.

Uso de políticas de IAM para Amazon Kinesis Data Streams y Amazon DynamoDB

La primera vez que habilita Amazon Kinesis Data Streams para Amazon DynamoDB, DynamoDB le crea automáticamente un AWS Identity and Access Management (IAM) vinculado a servicio. Esta función, `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`, permite a DynamoDB administrar la replicación de cambios a nivel de elemento en Kinesis Data Streams en su nombre. No elimine este rol vinculado a un servicio.

Para obtener más información acerca los roles vinculados a servicios, consulte Uso de roles vinculados a servicios en la Guía del usuario de IAM.

> **Note**
>
> DynamoDB no admite condiciones basadas en etiquetas para las políticas de IAM.

Para habilitar Amazon Kinesis Data Streams para Amazon DynamoDB, debe tener los siguientes permisos en la tabla:

- `dynamodb:EnableKinesisStreamingDestination`
- `kinesis:ListStreams`
- `kinesis:PutRecords`
- `kinesis:DescribeStream`

Para describir Amazon Kinesis Data Streams para Amazon DynamoDB para una tabla de DynamoDB determinada, debe tener los siguientes permisos en la tabla.

- `dynamodb:DescribeKinesisStreamingDestination`
- `kinesis:DescribeStreamSummary`
- `kinesis:DescribeStream`

Para deshabilitar Amazon Kinesis Data Streams para Amazon DynamoDB, debe tener los siguientes permisos en la tabla.

- `dynamodb:DisableKinesisStreamingDestination`

Para actualizar Amazon Kinesis Data Streams para Amazon DynamoDB, la tabla debe tener los siguientes permisos.

- `dynamodb:UpdateKinesisStreamingDestination`

En los siguientes ejemplos se muestra cómo utilizar las políticas de IAM para conceder permisos a Amazon Kinesis Data Streams para Amazon DynamoDB.

Ejemplo: Habilitar Amazon Kinesis Data Streams para Amazon DynamoDB

La siguiente política de IAM otorga permisos para activar Amazon Kinesis Data Streams para Amazon DynamoDB para la tabla `Music`. No concede permisos para desactivar, actualizar ni describir Kinesis Data Streams para DynamoDB para la tabla `Music`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/kinesisreplication.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBKinesisDataStreamsReplication",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName":
 "kinesisreplication.dynamodb.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "dynamodb:EnableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

Ejemplo: Actualizar Amazon Kinesis Data Streams para Amazon DynamoDB

La siguiente política de IAM otorga permisos para actualizar Amazon Kinesis Data Streams para Amazon DynamoDB para la tabla `Music`. No concede permisos para activar, desactivar ni describir Amazon Kinesis Data Streams para Amazon DynamoDB para la tabla `Music`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

Ejemplo: Deshabilite Amazon Kinesis Data Streams para Amazon DynamoDB

La siguiente política de IAM otorga permisos para desactivar Amazon Kinesis Data Streams para Amazon DynamoDB para la tabla `Music`. No concede permisos para activar, actualizar ni describir Amazon Kinesis Data Streams para Amazon DynamoDB para la tabla `Music`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        }
    ]
}
```

Ejemplo: aplicar permisos de forma selectiva para Amazon Kinesis Data Streams para Amazon DynamoDB basados en recursos

La siguiente política de IAM concede permisos para activar y describir Amazon Kinesis Data Streams para Amazon DynamoDB para la tabla `Music` y deniega permisos para desactivar Amazon Kinesis Data Streams para Amazon DynamoDB para la tabla `Orders`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:EnableKinesisStreamingDestination",
                "dynamodb:DescribeKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Music"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:DisableKinesisStreamingDestination"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Orders"
        }
    ]
}
```

Uso de roles vinculados a servicios para Kinesis Data Streams para DynamoDB

Amazon Kinesis Data Streams para Amazon DynamoDB utiliza roles vinculados a servicios de AWS Identity and Access Management (IAM). Un rol vinculado a un servicio es un tipo único de rol de IAM que está vinculado directamente a Kinesis Data Streams para DynamoDB. Los roles vinculados a servicios están predefinidos por Kinesis Data Streams para DynamoDB e incluyen todos los permisos que el servicio requiere para llamar a otros servicios de AWS en su nombre.

Un rol vinculado a un servicio simplifica la configuración de Kinesis Data Streams para DynamoDB porque ya no tendrá que agregar manualmente los permisos necesarios. Kinesis Data Streams para DynamoDB define los permisos de sus roles vinculados a servicios y, a menos que esté definido de otra manera, solo Kinesis Data Streams puede asumir sus roles. Los permisos definidos incluyen las políticas de confianza y de permisos, y que la política de permisos no se pueda asociar a ninguna otra entidad de IAM.

Para obtener información acerca de otros servicios que admiten roles vinculados a servicios, consulte Servicios de AWS que funcionan con IAM y busque los servicios que muestran Sí en la columna Rol vinculado a un servicio. Elija una opción Sí con un enlace para ver la documentación acerca del rol vinculado al servicio en cuestión.

Permisos de roles vinculados a servicios para Kinesis Data Streams para DynamoDB

Kinesis Data Streams para DynamoDB utiliza el rol vinculado a servicio denominado AWSServiceRoleForDynamodBkinesisDataStreamsReplication. El propósito del rol vinculado a servicios es permitir que Amazon DynamoDB administre la replicación de cambios a nivel de elemento en Kinesis Data Streams, en su nombre.

El rol vinculado al servicio `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication` depende de los siguientes servicios para asumir el rol:

- `kinesisreplication.dynamodb.amazonaws.com`

La política de permisos del rol permite que Kinesis Data Streams para DynamoDB realice las siguientes acciones en los recursos especificados:

- Acción: `Put records and describe` en `Kinesis stream`
- Acción: `Generate data keys` en `AWS KMS` para poner datos en las transmisiones de Kinesis cifrados mediante claves de AWS KMS generadas por el usuario.

Para conocer el contenido exacto del documento de política, consulte DynamoDBKinesisReplicationServiceRolePolicy.

Debe configurar permisos para permitir a una entidad de IAM (como un usuario, grupo o rol) crear, editar o eliminar un rol vinculado a servicios. Para obtener más información, consulte Permisos de roles vinculados a servicios en la Guía del usuario de IAM.

Creación de un rol vinculado a un servicio para Kinesis Data Streams para DynamoDB

No necesita crear manualmente un rol vinculado a servicios. Cuando habilita Kinesis Data Streams para DynamoDB en la Consola de administración de AWS, la AWS CLI, o la API de AWS, Kinesis Data Streams para DynamoDB crea automáticamente el rol vinculado al servicio.

Si elimina este rol vinculado a servicios y necesita crearlo de nuevo, puede utilizar el mismo proceso para volver a crear el rol en su cuenta. Cuando habilita Kinesis Data Streams para DynamoDB, Kinesis Data Streams para DynamoDB vuelve a crear el rol vinculado al servicio.

Edición de un rol vinculado a un servicio para Kinesis Data Streams para DynamoDB

Kinesis Data Streams para DynamoDB no permite editar el rol vinculado al servicio `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`. Después de crear un rol vinculado al servicio, no podrá cambiar el nombre del rol, ya que varias entidades podrían hacer referencia al rol. Sin embargo, sí puede editar la descripción del rol con IAM. Para obtener más información, consulte Edición de un rol vinculado a servicios en la Guía del usuario de IAM.

Eliminación de un rol vinculado a un servicio para Kinesis Data Streams para DynamoDB

También puede utilizar la consola de IAM, la AWS CLI o la API de AWS para eliminar manualmente el rol vinculado al servicio. Para ello, primero debe limpiar manualmente los recursos del rol vinculado al servicio para poder eliminarlo después manualmente.

> **Note**
>
> Si el servicio Kinesis Data Streams para DynamoDB está utilizando el rol cuando intenta eliminar los recursos, la eliminación podría producir un error. En tal caso, espere unos minutos e intente de nuevo la operación.

Para eliminar manualmente el rol vinculado a servicios mediante IAM

Puede usar la consola de IAM, la AWS CLI o la API de AWS para eliminar el rol vinculado a un servicio de `AWSServiceRoleForDynamoDBKinesisDataStreamsReplication`. Para más información, consulte Eliminación de un rol vinculado a servicios en la Guía del usuario de IAM.

Captura de datos de cambios para DynamoDB Streams

DynamoDB Streams captura una secuencia en orden cronológico de las modificaciones de los elementos en una tabla de DynamoDB y almacena esta información en un log durante un máximo de 24 horas. Las aplicaciones pueden obtener acceso a este registro y ver los elementos de datos tal y como se encontraban antes y después de la modificación, prácticamente en tiempo real.

El cifrado en reposo cifra los datos en DynamoDB streams. Para obtener más información, consulte Cifrado en reposo en DynamoDB.

Una transmisión de DynamoDB es un flujo ordenado de información sobre los cambios que se realizan en los elementos de una tabla de DynamoDB. Cuando se habilita una transmisión en una tabla, DynamoDB obtiene información sobre cada modificación de los elementos de datos de esa tabla.

Cada vez que una aplicación crea, actualiza o elimina elementos en la tabla, DynamoDB Streams escribe un registro de transmisión con los atributos de clave principal de los elementos modificados. Un registro de transmisión contiene información sobre una modificación de los datos de un solo elemento de una tabla de DynamoDB. Puede configurar la secuencia de tal forma que sus registros capturen información adicional; por ejemplo, las imágenes de "antes" y "después" de los elementos modificados.

DynamoDB Streams ayuda a garantizar lo siguiente:

- Cada registro de secuencia aparece una única vez en la secuencia.
- Para cada elemento que se modifica de una tabla de DynamoDB, los registros de transmisión aparecen en el mismo orden en que se han realizado las modificaciones del elemento.

DynamoDB Streams escribe los registros de transmisión prácticamente en tiempo real, para que pueda crear aplicaciones que consuman estas transmisiones y adopten medidas en función de su contenido.

Temas

- Endpoints para DynamoDB Streams

- Habilitación de una secuencia
- Lectura y procesamiento de un flujo
- DynamoDB Streams y período de vida
- Uso del adaptador Kinesis de DynamoDB Streams para procesar registros de transmisión
- API de bajo nivel de DynamoDB Streams: ejemplo en Java
- DynamoDB Streams y disparadores de AWS Lambda
- DynamoDB Streams y Apache Flink

Endpoints para DynamoDB Streams

AWS mantiene puntos de enlace distintos para DynamoDB y DynamoDB Streams. Para usar las tablas y los índices de la base de datos, la aplicación tendrá que acceder a un punto de enlace de DynamoDB. Para leer y procesar los registros de DynamoDB Streams, la aplicación tendrá que obtener acceso a un punto de enlace de DynamoDB Streams situado en la misma región.

DynamoDB Streams ofrece dos conjuntos de puntos de conexión. Son los siguientes:

- Puntos de conexión exclusivos para IPv4: puntos de conexión con la convención de nomenclatura de `streams.dynamodb.<region>.amazonaws.com`.
- Puntos de conexión de doble pila: puntos de conexión nuevos que son compatibles con IPv4 e IPv6 y que siguen la convención de nomenclatura de `streams-dynamodb.<region>.api.aws`.

> **Note**
>
> Para obtener una lista completa de las regiones y puntos de conexión de DynamoDB y DynamoDB Streams, consulte Regiones y puntos de conexión en la Referencia general de AWS.

Los SDK de AWS proporcionan clientes independientes para DynamoDB y DynamoDB Streams. Según cuáles sean sus necesidades, la aplicación puede obtener acceso a un punto de enlace de DynamoDB, a un punto de enlace de DynamoDB Streams o a ambos al mismo tiempo. Para conectarse a ambos puntos de enlace, la aplicación tendrá que crear instancias de dos clientes: uno para DynamoDB y otro para DynamoDB Streams.

Habilitación de una secuencia

Puede habilitar una transmisión en una nueva tabla al crearla mediante la AWS CLI o uno de los SDK de AWS. También puede habilitar o deshabilitar una transmisión en una tabla existente, así como cambiar la configuración de una transmisión. DynamoDB Streams opera de forma asincrónica, por lo que el rendimiento de una tabla no se vea afectado al habilitar una transmisión.

La forma más sencilla de administrar DynamoDB Streams es usar la Consola de administración de AWS.

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de la consola de DynamoDB, elija Tablas y seleccione una tabla existente.
3. Elija la pestaña Exports and streams (Exportaciones y flujos).
4. En la sección Detalles del flujo de DynamoDB, seleccione Activar.
5. En la ventana Activar el flujo de DynamoDB, elija la información que se escribirá en el flujo cada vez que se modifiquen los datos de la tabla:

   - Key attributes only (Solo atributos clave): solo los atributos clave del elemento modificado.
   - New image (Nueva imagen): el elemento completo tal y como aparece después de modificarlo.
   - Old image (Imagen anterior): el elemento completo tal y como aparecía antes de modificarlo.
   - New and old images (Imágenes nuevas y anteriores): ambas imágenes del elemento, la nueva y la anterior.

   Cuando la configuración sea la que desea, elija Activar flujo.
6. (Opcional) Para desactivar un flujo existente, elija Desactivar bajo Detalles del flujo de DynamoDB.

También puede usar las operaciones de la API `CreateTable` o `UpdateTable` para habilitar o modificar una secuencia, respectivamente. El parámetro `StreamSpecification` determina cómo se configura la secuencia:

- `StreamEnabled`: especifica si una transmisión está habilitada (`true`) o deshabilitada (`false`) para la tabla.
- `StreamViewType`: especifica la información que se escribirá en la transmisión cada vez que se modifiquen los datos de la tabla:

- `KEYS_ONLY`: solo los atributos de clave del elemento modificado.
- `NEW_IMAGE`: el elemento completo tal y como aparece después de modificarlo.
- `OLD_IMAGE`: el elemento completo tal y como aparecía antes de modificarlo.
- `NEW_AND_OLD_IMAGES`: ambas imágenes del elemento, la nueva y la anterior.

Puede habilitar o deshabilitar una secuencia en cualquier momento. Sin embargo, tenga en cuenta que recibirá una excepción `ValidationException` si intenta habilitar una secuencia en una tabla que ya tiene una. Recibirá una `ValidationException` si intenta desactivar una secuencia en una tabla que no tiene ninguna.

Cuando se establece `StreamEnabled` en `true`, DynamoDB crea una nueva transmisión con un descriptor de transmisión único asignado a ella. Si deshabilita y vuelve a habilitar una secuencia en la tabla, se crea una secuencia nueva con un descriptor de secuencia distinto.

Cada secuencia se identifica de forma exclusiva mediante un nombre de recurso de Amazon (ARN). A continuación se muestra un ejemplo de ARN de una transmisión de una tabla de DynamoDB denominada `TestTable`.

```
arn:aws:dynamodb:us-west-2:111122223333:table/TestTable/stream/2015-05-11T21:21:33.291
```

Para determinar el último descriptor de transmisión de una tabla, se emite una solicitud `DescribeTable` de DynamoDB y se busca el elemento `LatestStreamArn` en la respuesta.

> **Note**
>
> No es posible editar un `StreamViewType` una vez que se ha configurado un flujo. Si necesita realizar cambios en un flujo después de haberlo configurado, debe desactivar el flujo actual y crear uno nuevo.

Lectura y procesamiento de un flujo

Para leer y procesar una transmisión, la aplicación tiene que conectarse a un punto de enlace de DynamoDB Streams y emitir solicitudes de API.

Una secuencia consta de registros de secuencia. Cada registro de transmisión representa una única modificación de datos de la tabla de DynamoDB a la que pertenece la secuencia. A cada registro

de secuencia se le asigna un número de secuencia que refleja el orden en que el registro se ha publicado en la secuencia.

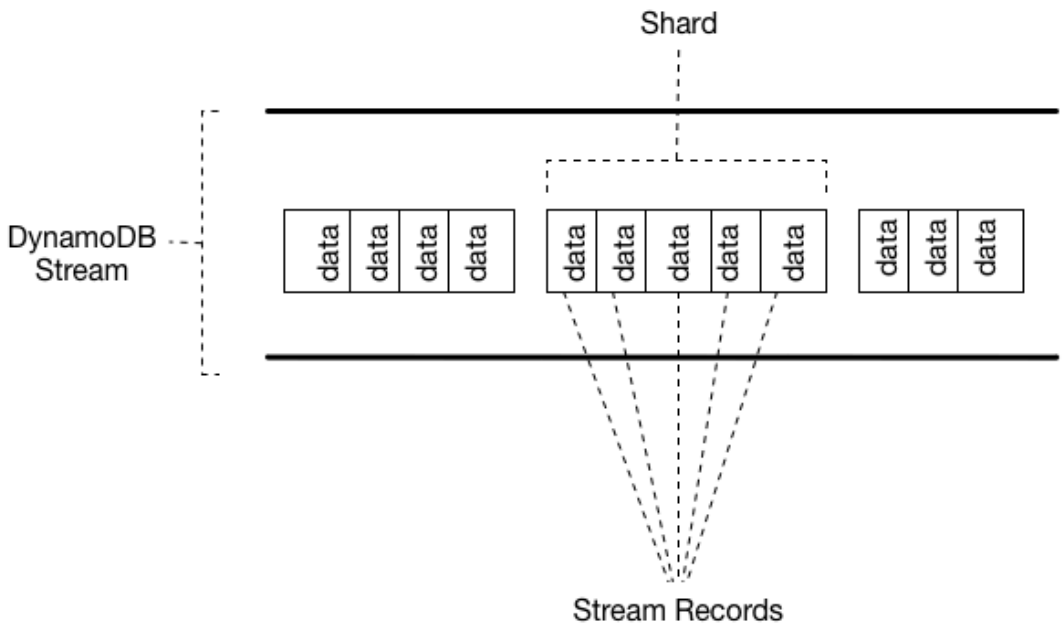
Los registros de secuencia se organizan en grupos, o fragmentos. Cada fragmento actúa como contenedor de varios registros de secuencia y contiene la información necesaria para obtener acceso a estos registros y recorrerlos en iteración. Los registros de secuencia de un fragmento se eliminan automáticamente transcurridas de 24 horas.

Los fragmentos son efímeros: se crean y eliminan automáticamente, según sea necesario. Además, cualquier fragmento se puede dividir en varios fragmentos nuevos; esto también sucede automáticamente. (Cabe destacar que un fragmento principal puede tener un solo fragmento secundario). Un fragmento se puede dividir en respuesta a niveles de actividad de escritura elevados en la tabla principal, para que las aplicaciones puedan procesar en paralelo los registros de varios fragmentos.

Si se deshabilita una secuencia, todos fragmentos que estén abiertos se cerrarán. Los datos de la transmisión continuarán disponibles para leerlos durante 24 horas.

Como los fragmentos poseen un parentesco (principales y secundarios), las aplicaciones siempre deben procesar los fragmentos principales antes de procesar los secundarios. Esto ayuda a garantizar que los registros de secuencia se procesen también en el orden correcto. (Si utiliza DynamoDB Streams Kinesis Adapter, esto se lleva a cabo de forma automatizada: la aplicación procesará los fragmentos y registros de transmisión en el orden correcto y también administrará automáticamente los fragmentos nuevos o vencidos, así como aquellos que se dividan mientras se ejecuta la aplicación. Para obtener más información, consulte). Uso del adaptador Kinesis de DynamoDB Streams para procesar registros de transmisión.)

En el siguiente diagrama se muestra la relación entre una secuencia, sus fragmentos y los registros de secuencia contenidos en los fragmentos.



> **Note**
>
> Si lleva a cabo una operación `PutItem` o `UpdateItem` que no modifica ningún dato de un elemento, DynamoDB Streams no escribe ningún registro de transmisión de esa operación.

Para obtener acceso a una secuencia y procesar los registros que contiene, proceda como sigue:

- Determine el ARN único de la secuencia a la que desea obtener acceso.
- Determine qué fragmentos de la secuencia contienen los registros de secuencia que le interesan.
- Obtenga acceso a los fragmentos y recupere los registros de secuencia que desee.

> **Note**
>
> Nunca debe haber más de dos procesos leyendo la misma partición de flujo a la vez. Usar más de dos procesos de lectura por fragmento puede provocar que se aplique la limitación controlada.

La API de DynamoDB Streams ofrece las siguientes acciones para usarlas en los programas de aplicación:

- `ListStreams`: devuelve una lista de descriptores de transmisión de la cuenta y el punto de enlace actuales. Si lo desea, puede solicitar únicamente los descriptores de secuencia de un nombre de tabla concreto.
- `DescribeStream`: devuelve información sobre un flujo, incluido el estado actual del flujo, su nombre de recurso de Amazon (ARN), la composición de sus particiones y su tabla de DynamoDB correspondiente. Puede utilizar opcionalmente el campo `ShardFilter` para recuperar la partición secundaria existente asociada a la partición principal.
- `GetShardIterator`: devuelve un iterador de fragmentos que describe una ubicación en el fragmento. Puede solicitar que el iterador proporcione acceso al punto más antiguo, al punto más reciente o a un punto concreto de la secuencia.
- `GetRecords`: devuelve los registros de secuencia de un fragmento determinado. Debe proporcionar el iterador de fragmentos devuelto por una solicitud `GetShardIterator`.

Para obtener descripciones completas de estas operaciones de la API, así como ejemplos de solicitudes y respuestas, consulte la Referencia de API de Amazon DynamoDB Streams.

Detección de particiones

Descubra nuevas particiones en el flujo de DynamoDB con dos métodos potentes. Como usuario de Amazon DynamoDB Streams, dispone de dos formas efectivas de rastrear e identificar nuevas particiones:

Sondeo de toda la topología del flujo

Use la API de `DescribeStream` para sondear el flujo con regularidad. Esto devuelve todas las particiones de la transmisión, incluidas las nuevas particiones que se hayan creado. Al comparar los resultados a lo largo del tiempo, puede detectar las particiones recién agregadas.

Detección de particiones secundarias

Use la API de `DescribeStream` con el parámetro `ShardFilter` para encontrar un subconjunto de particiones. Al especificar una partición principal en la solicitud, los flujos de DynamoDB devolverán sus particiones secundarias inmediatas. Este enfoque resulta útil cuando solo se necesita rastrear el linaje de las particiones sin escanear todo el flujo.

Las aplicaciones que consumen datos de los flujos de DynamoDB pueden pasar de manera eficiente de leer una partición cerrada a su partición secundaria mediante este parámetro `ShardFilter`, lo que evita llamadas repetidas a la API de `DescribeStream` para recuperar

y recorrer el mapa de particiones de todas las particiones cerradas y abiertas. Esto ayuda a detectar rápidamente las particiones secundarias después de cerrar la partición principal, lo que hace que las aplicaciones de procesamiento de flujos sean más receptivas y rentables.

Ambos métodos le permiten mantenerse al tanto de la estructura en evolución de los flujos de DynamoDB, lo que garantiza que nunca se pierda las actualizaciones de datos importantes ni las modificaciones de las particiones.

Límite de retención de datos para DynamoDB Streams

Todos los datos de DynamoDB Streams están sujetos a una vida útil de 24 horas. Puede recuperar y analizar las últimas 24 horas de actividad de cada tabla. Sin embargo, los datos de más de 24 horas se pueden recortar (eliminar) en cualquier momento.

Si deshabilita una secuencia en una tabla, los datos de esa secuencia continuarán disponibles para leerlos durante 24 horas. Transcurrido ese tiempo, los datos vencen y los registros de secuencia se eliminan automáticamente. No existe ningún mecanismo para eliminar manualmente las secuencias. Tiene que esperar hasta que se alcance el límite de retención (24 horas) para que se eliminen los registros de secuencia.

DynamoDB Streams y período de vida

Puede realizar copias de seguridad o bien procesar los elementos eliminados por período de vida (TTL, por sus siglas en inglés) habilitando Amazon DynamoDB Streams en la tabla y procesando los registros de transmisión de los elementos vencidos. Para obtener más información, consulte Lectura y procesamiento de un flujo.

El registro de secuencias contiene el campo de identidad del usuario `Records[<index>].userIdentity`.

Los elementos que el proceso período de vida elimina tras su vencimiento tienen los siguientes campos:

- `Records[<index>].userIdentity.type`

  `"Service"`
- `Records[<index>].userIdentity.principalId`

  `"dynamodb.amazonaws.com"`

> **Note**
>
> Cuando utilice el TTL en una tabla global, el campo `userIdentity` se configurará en la región en la que se realizó el TTL. Este campo no se establecerá en otras regiones cuando se replique la eliminación.

En el código JSON siguiente se muestra la parte pertinente de un registro de secuencias único.

```
"Records": [
    {
        ...

        "userIdentity": {
            "type": "Service",
            "principalId": "dynamodb.amazonaws.com"
        }

        ...

    }
]
```

Uso de DynamoDB Streams y Lambda para archivar elementos de TTL eliminados

La combinación de Periodo de vida (TTL) de DynamoDB, DynamoDB Streams y AWS Lambda puede simplificar el archivo de datos, reducir los costos de almacenamiento de DynamoDB y reducir la complejidad del código. El uso de Lambda como consumidor de flujos proporciona muchas ventajas, entre las que destaca la reducción de costos en comparación con otros consumidores como Kinesis Client Library (KCL). No se le cobran las llamadas a la API `GetRecords` en su flujo de DynamoDB cuando utiliza Lambda para consumir eventos, y Lambda puede proporcionar un filtrado de eventos mediante la identificación de patrones JSON en un evento de flujo. Con el filtrado de contenido de patrones de eventos, puede definir hasta cinco filtros diferentes para controlar qué eventos se envían a Lambda para procesarlos. De este modo, se reducen las invocaciones de sus funciones Lambda, se simplifica el código y se reduce el costo total.

Aunque DynamoDB Streams contiene todas las modificaciones de datos, como las acciones `Create`, `Modify` y `Remove`, esto puede dar lugar a invocaciones no deseadas de su función Lambda de archivo. Por ejemplo, supongamos que tiene una tabla con dos millones de

modificaciones de datos por hora que se incluyen en el flujo, pero menos del 5 por ciento de estas son eliminaciones de elementos que vencerán a través del proceso TTL y se deben archivar. Con los filtros de origen de eventos de Lambda, la función Lambda solo se invocará 100 000 veces por hora. El resultado con el filtrado de eventos es que solo se le cobran las invocaciones necesarias en lugar de los dos millones de invocaciones que tendría sin el filtrado de eventos.

El filtrado de eventos se aplica a la asignación del origen de eventos de Lambda, que es un recurso que lee de un evento elegido (el flujo de DynamoDB) e invoca una función Lambda. En el siguiente diagrama, puede ver cómo una función Lambda consume un elemento de TTL eliminado mediante flujos y filtros de eventos.

Patrón de filtros de eventos de DynamoDB Time to Live

Si agrega el siguiente JSON a sus criterios de filtro de asignación de origen de eventos, podrá invocar su función Lambda solo para los elementos de TTL eliminados:

```
{
    "Filters": [
        {
            "Pattern": { "userIdentity": { "type": ["Service"], "principalId":
 ["dynamodb.amazonaws.com"] } }
        }
    ]
}
```

Creación de una asignación de origen de eventos de AWS Lambda

Utilice los siguientes fragmentos de código para crear una asignación de origen de eventos filtrados que pueda conectar con el flujo de DynamoDB de una tabla. Cada bloque de código incluye el patrón del filtro de eventos.

AWS CLI

```
aws lambda create-event-source-mapping \
--event-source-arn 'arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000' \
```

```
--batch-size 10 \
--enabled \
--function-name test_func \
--starting-position LATEST \
--filter-criteria '{"Filters": [{"Pattern": "{\"userIdentity\":{\"type\":[\"Service
\"],\"principalId\":[\"dynamodb.amazonaws.com\"]}}"}]}'
```

Java

```
LambdaClient client = LambdaClient.builder()
        .region(Region.EU_WEST_1)
        .build();

Filter userIdentity = Filter.builder()
        .pattern("{\"userIdentity\":{\"type\":[\"Service\"],\"principalId\":
[\"dynamodb.amazonaws.com\"]}}")
        .build();

FilterCriteria filterCriteria = FilterCriteria.builder()
        .filters(userIdentity)
        .build();

CreateEventSourceMappingRequest mappingRequest =
 CreateEventSourceMappingRequest.builder()
        .eventSourceArn("arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000")
        .batchSize(10)
        .enabled(Boolean.TRUE)
        .functionName("test_func")
        .startingPosition("LATEST")
        .filterCriteria(filterCriteria)
        .build();

try{
    CreateEventSourceMappingResponse eventSourceMappingResponse =
 client.createEventSourceMapping(mappingRequest);
    System.out.println("The mapping ARN is
 "+eventSourceMappingResponse.eventSourceArn());

}catch (ServiceException e){
    System.out.println(e.getMessage());
}
```

Node

```
const client = new LambdaClient({ region: "eu-west-1" });

const input = {
    EventSourceArn: "arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000",
    BatchSize: 10,
    Enabled: true,
    FunctionName: "test_func",
    StartingPosition: "LATEST",
    FilterCriteria: { "Filters": [{ "Pattern": "{\"userIdentity\":{\"type\":
[\"Service\"],\"principalId\":[\"dynamodb.amazonaws.com\"]}}" }] }
}

const command = new CreateEventSourceMappingCommand(input);

try {
    const results = await client.send(command);
    console.log(results);
} catch (err) {
    console.error(err);
}
```

Python

```
session = boto3.session.Session(region_name = 'eu-west-1')
client = session.client('lambda')

try:
    response = client.create_event_source_mapping(
        EventSourceArn='arn:aws:dynamodb:eu-west-1:012345678910:table/test/
stream/2021-12-10T00:00:00.000',
        BatchSize=10,
        Enabled=True,
        FunctionName='test_func',
        StartingPosition='LATEST',
        FilterCriteria={
            'Filters': [
                {
                    'Pattern': "{\"userIdentity\":{\"type\":[\"Service\"],
\"principalId\":[\"dynamodb.amazonaws.com\"]}}"
                },
```

```
            ]
        }
    )
    print(response)
except Exception as e:
    print(e)
```

JSON

```
{
  "userIdentity": {
     "type": ["Service"],
     "principalId": ["dynamodb.amazonaws.com"]
   }
}
```

Uso del adaptador Kinesis de DynamoDB Streams para procesar registros de transmisión

Usar Amazon Kinesis Adapter es la forma recomendada de consumir secuencias de Amazon DynamoDB. La API de DynamoDB Streams es intencionadamente similar a la de Kinesis Data Streams. En ambos servicios, los data streams se componen de fragmentos, que son los contenedores de los registros de secuencia. Los API de ambos servicios contienen operaciones `ListStreams`, `DescribeStream`, `GetShards` y `GetShardIterator`. (Aunque estas acciones de DynamoDB Streams son parecidas a sus homólogas de Kinesis Data Streams, no son idénticas al 100 %).

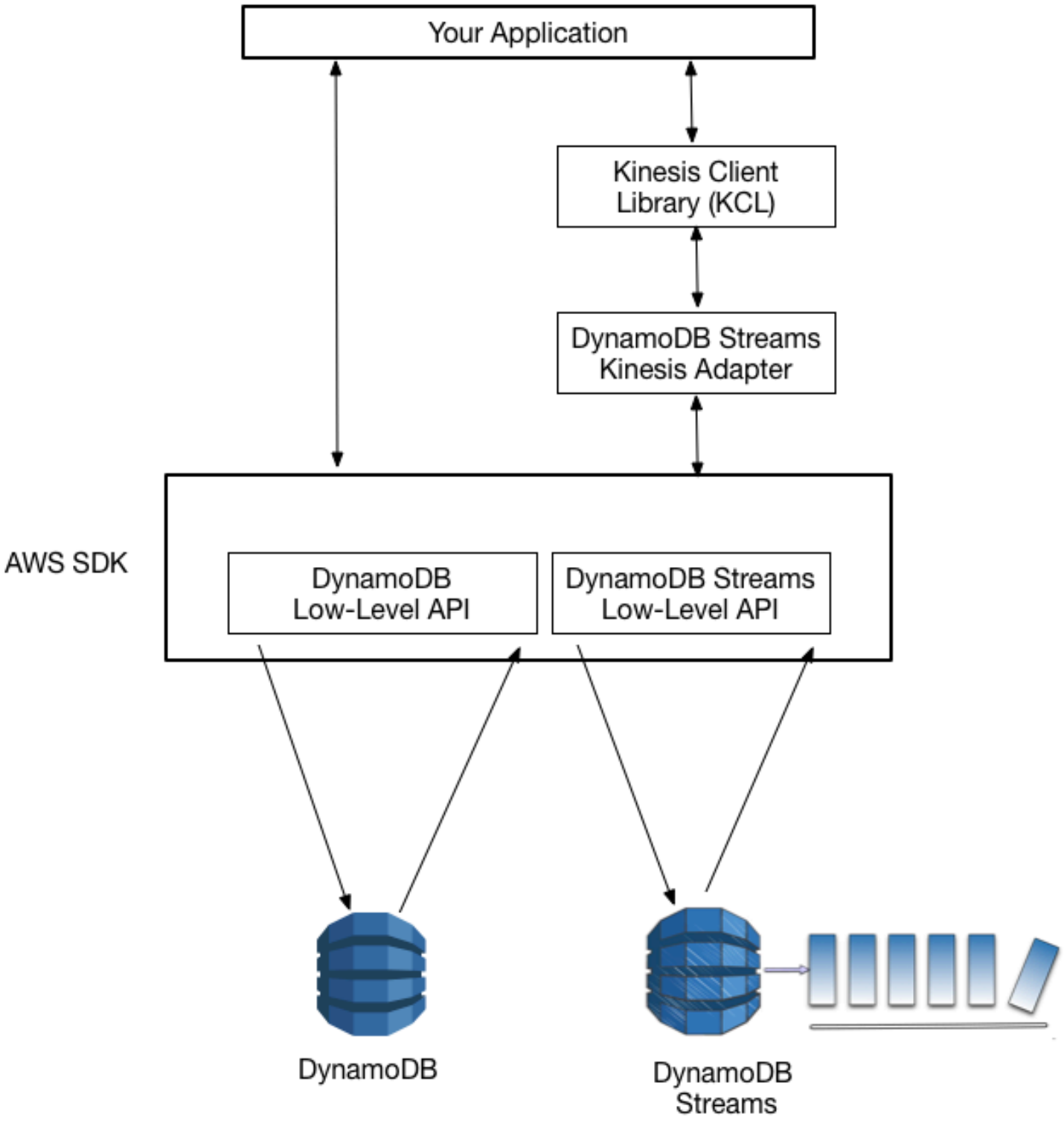
Como usuario de DynamoDB Streams, puede sacar partido de los patrones de diseño contenidos en KCL para procesar los fragmentos de DynamoDB Streams y transmitir registros. Para ello, se utiliza DynamoDB Streams Kinesis Adapter. Kinesis Adapter implementa la interfaz de Kinesis Data Streams, de tal forma que se pueda usar KCL para consumir y procesar registros desde DynamoDB Streams. Para obtener instrucciones acerca de cómo configurar e instalar DynamoDB Streams Kinesis Adapter, consulte el repositorio de GitHub.

Puede escribir aplicaciones para Kinesis Data Streams mediante Kinesis Client Library (KCL). KCL simplifica la codificación porque proporciona abstracciones útiles por encima del API de bajo nivel de Kinesis Data Streams. Para obtener más información sobre KCL, consulte Desarrollo de consumidores mediante la biblioteca Kinesis Client Library en la Guía para desarrolladores de Amazon Kinesis Data Streams.

DynamoDB recomienda utilizar la versión 3.x de KCL con AWS SDK para Java v2.x. La versión actual del adaptador de Kinesis de DynamoDB Streams 1.x con AWS SDK para AWS SDK para Java v1.x se seguirá admitiendo por completo durante todo el ciclo de vida, tal y como estaba previsto durante el periodo de transición, en consonancia con la Política de mantenimiento de AWS SDK y herramientas.

> **Note**
>
> Las versiones 1.x y 2.x de la biblioteca de clientes de Amazon Kinesis (KCL) están obsoletas. KCL 1.x dejará de recibir asistencia el 30 de enero de 2026. Le recomendamos que migre las aplicaciones de KCL que utilizan la versión 1.x a la última versión de KCL antes del 30 de enero de 2026. Para encontrar la versión más reciente de KCL, consulte la página de la Biblioteca de clientes de Amazon Kinesis en GitHub. Para obtener información sobre las últimas versiones de KCL, consulte Uso de la biblioteca de clientes de Kinesis. Para obtener información sobre cómo migrar de KCL 1.x a KCL 3.x, consulte Migración de KCL 1.x a KCL 3.x.

En el siguiente diagrama se muestra cómo interaccionan estas bibliotecas entre sí.



Si DynamoDB Streams Kinesis Adapter está implementado, puede comenzar a desarrollar para la interfaz de KCL y dirigir las llamadas al API de forma transparente al punto de enlace de DynamoDB Streams.

Cuando se inicia la aplicación, llama a KCL para crear una instancia de un proceso de trabajo. Debe facilitar al proceso de trabajo información sobre la configuración de la aplicación, como el descriptor de la transmisión y las credenciales de AWS, así como el nombre de una clase de procesador de registros que usted proporcione. A medida que el proceso de trabajo ejecuta el código en el procesador de registros, lleva a cabo las siguientes tareas:

- Se conecta a la secuencia
- Enumera las particiones del flujo.
- Comprueba y enumera las particiones secundarias de una partición principal cerrada dentro del flujo
- Coordina la asociación de los fragmentos con otros procesos de trabajo (si procede)
- Crea instancias de un procesador de registros para cada fragmento que administra
- Extrae registros del flujo.
- Inserta los registros en el procesador de registros correspondiente
- Genera puntos de comprobación para los registros procesados
- Balancea las asociaciones entre fragmentos y procesos de trabajo cuando cambia el recuento de instancias de procesos de trabajo
- Equilibra las asociaciones entre particiones y procesos de trabajo cuando las particiones se dividen.

> **Note**
>
> Para obtener una descripción de los conceptos de KCL enumerados aquí, consulte Desarrollo de consumidores mediante la biblioteca Kinesis Client Library en la Guía para desarrolladores de Amazon Kinesis Data Streams.
> Para obtener más información acerca de cómo usar los flujos con AWS Lambda, consulte DynamoDB Streams y disparadores de AWS Lambda.

Migración de KCL 1.x a KCL 3.x

Descripción general

En esta guía se proporcionan instrucciones para migrar la aplicación de consumidor de KCL 1.x a KCL 3.x. Debido a las diferencias de arquitectura entre KCL 1.x y KCL 3.x, la migración requiere actualizar varios componentes para garantizar la compatibilidad.

KCL 1.x utiliza clases e interfaces diferentes en comparación con KCL 3.x. Debe migrar primero el procesador de registros, el generador de procesadores de registros y las clases de procesos de trabajo al formato compatible con KCL 3.x y, a continuación, seguir los pasos de migración de KCL 1.x a KCL 3.x.

Pasos para realizar la migración

Temas

- Paso 1: migración del procesador de registros
- Paso 2: migración del generador de procesadores de registros
- Paso 3: migración del proceso de trabajo
- Paso 4: información general y recomendaciones sobre la configuración de KCL 3.x
- Paso 5: migración de KCL 2.x a KCL 3.x

Paso 1: migración del procesador de registros

En el siguiente ejemplo se muestra un procesador de registros implementado para el adaptador de Kinesis de DynamoDB Streams de KCL 1.x:

```
package com.amazonaws.kcl;

import com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ShutdownInput;

public class StreamsRecordProcessor implements IRecordProcessor,
 IShutdownNotificationAware {
    @Override
    public void initialize(InitializationInput initializationInput) {
        //
        // Setup record processor
        //
```

```
    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        for (Record record : processRecordsInput.getRecords()) {
            String data = new String(record.getData().array(),
 Charset.forName("UTF-8"));
            System.out.println(data);
            if (record instanceof RecordAdapter) {
                // record processing and checkpointing logic
            }
        }
    }

    @Override
    public void shutdown(ShutdownInput shutdownInput) {
        if (shutdownInput.getShutdownReason() == ShutdownReason.TERMINATE) {
            try {
                shutdownInput.getCheckpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                throw new RuntimeException(e);
            }
        }
    }

    @Override
    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
        try {
            checkpointer.checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow exception
            //
            e.printStackTrace();
        }
    }
}
```

Migración de la clase RecordProcessor

1. Cambie las interfaces de `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor` y

`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificat`
a
`software.amazon.dynamodb.streamsadapter.processor.DynamoDBStreamsShardRecordP`
tal y como se indica a continuación:

```
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;

import
 software.amazon.dynamodb.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
```

2. Actualice las instrucciones de importación para los métodos `initialize` y `processRecords`:

```
// import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import software.amazon.kinesis.lifecycle.events.InitializationInput;

// import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
```

3. Sustituya el método `shutdownRequested` por los métodos nuevos siguientes: `leaseLost`, `shardEnded` y `shutdownRequested`.

```
//    @Override
//    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
//        //
//        // This is moved to shardEnded(...) and
 shutdownRequested(ShutdownReauestedInput)
//        //
//        try {
//            checkpointer.checkpoint();
//        } catch (ShutdownException | InvalidStateException e) {
//            //
//            // Swallow exception
//            //
//            e.printStackTrace();
//        }
//    }

    @Override
```

```
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
```

A continuación, se muestra la versión actualizada de la clase del procesador de registros:

```
package com.amazonaws.codesamples;

import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import
 software.amazon.dynamodb.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor;
```

```
import
 software.amazon.dynamodb.streamsadapter.adapter.DynamoDBStreamsKinesisClientRecord;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
import
 com.amazonaws.services.dynamodbv2.streamsadapter.adapter.DynamoDBStreamsClientRecord;
import software.amazon.awssdk.services.dynamodb.model.Record;

public class StreamsRecordProcessor implements DynamoDBStreamsShardRecordProcessor {

    @Override
    public void initialize(InitializationInput initializationInput) {

    }

    @Override
    public void processRecords(DynamoDBStreamsProcessRecordsInput processRecordsInput)
 {
        for (DynamoDBStreamsKinesisClientRecord record: processRecordsInput.records())
            Record ddbRecord = record.getRecord();
            // processing and checkpointing logic for the ddbRecord
        }
    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
```

```
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
}
```

> **Note**
>
> El adaptador de Kinesis de DynamoDB Streams ahora usa el modelo de registro de SDKv2. En SDKv2, los objetos `AttributeValue` complejos (`BS`, `NS`, `M`, `L` y `SS`) nunca devuelven un valor nulo. Use los métodos `hasBs()`, `hasNs()`, `hasM()`, `hasL()` y `hasSs()` para verificar si estos valores existen.

Paso 2: migración del generador de procesadores de registros

El generador de procesadores de registros es responsable de la creación de procesadores de registros cuando se adquiere una asignación. A continuación, se muestra un ejemplo de un generador de la versión 1.x de KCL:

```
package com.amazonaws.codesamples;

import software.amazon.dynamodb.AmazonDynamoDB;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;

public class StreamsRecordProcessorFactory implements IRecordProcessorFactory {

    @Override
    public IRecordProcessor createProcessor() {
        return new StreamsRecordProcessor(dynamoDBClient, tableName);
    }
}
```

Migración de **`RecordProcessorFactory`**

- Cambie la interfaz implementada de `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFa` a `software.amazon.kinesis.processor.ShardRecordProcessorFactory`, tal y como se indica a continuación:

```
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessor;

// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

// public class TestRecordProcessorFactory implements IRecordProcessorFactory {
public class StreamsRecordProcessorFactory implements ShardRecordProcessorFactory {

Change the return signature for createProcessor.

// public IRecordProcessor createProcessor() {
public ShardRecordProcessor shardRecordProcessor() {
```

A continuación, se muestra un ejemplo de generador de procesadores de registros de la versión 3.0:

```
package com.amazonaws.codesamples;

import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class StreamsRecordProcessorFactory implements ShardRecordProcessorFactory {

    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new StreamsRecordProcessor();
    }
}
```

Paso 3: migración del proceso de trabajo

En la versión 3.0 de KCL, una nueva clase, llamada Scheduler, reemplaza la clase Worker. A continuación, se muestra un ejemplo de proceso de trabajo de la versión 1.x de KCL:

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = StreamsWorkerFactory.createDynamoDbStreamsWorker(
        recordProcessorFactory,
        workerConfig,
        adapterClient,
        amazonDynamoDB,
        amazonCloudWatchClient);
```

Para migrar el proceso de trabajo

1. Cambie la instrucción `import` para la clase `Worker` por las instrucciones de importación para las clases `Scheduler` y `ConfigsBuilder`.

   ```
   // import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
   import software.amazon.kinesis.coordinator.Scheduler;
   import software.amazon.kinesis.common.ConfigsBuilder;
   ```

2. Importe `StreamTracker` y cambie la importación de `StreamsWorkerFactory` a `StreamsSchedulerFactory`.

   ```
   import software.amazon.kinesis.processor.StreamTracker;
   // import software.amazon.dynamodb.streamsadapter.StreamsWorkerFactory;
   import software.amazon.dynamodb.streamsadapter.StreamsSchedulerFactory;
   ```

3. Elija la posición desde la que desea iniciar la aplicación. Puede ser `TRIM_HORIZON` o `LATEST`.

   ```
   import software.amazon.kinesis.common.InitialPositionInStream;
   import software.amazon.kinesis.common.InitialPositionInStreamExtended;
   ```

4. Crear una instancia de `StreamTracker`.

   ```
   StreamTracker streamTracker = StreamsSchedulerFactory.createSingleStreamTracker(
           streamArn,

    InitialPositionInStreamExtended.newInitialPosition(InitialPositionInStream.TRIM_HORIZON)
   ```

```
);
```

5. Cree el objeto `AmazonDynamoDBStreamsAdapterClient`.

```
import software.amazon.dynamodb.streamsadapter.AmazonDynamoDBStreamsAdapterClient;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;

...

AwsCredentialsProvider credentialsProvider = DefaultCredentialsProvider.create();

AmazonDynamoDBStreamsAdapterClient adapterClient = new
 AmazonDynamoDBStreamsAdapterClient(
        credentialsProvider, awsRegion);
```

6. Cree el objeto `ConfigsBuilder`.

```
import software.amazon.kinesis.common.ConfigsBuilder;

...
ConfigsBuilder configsBuilder = new ConfigsBuilder(
                streamTracker,
                applicationName,
                adapterClient,
                dynamoDbAsyncClient,
                cloudWatchAsyncClient,
                UUID.randomUUID().toString(),
                new StreamsRecordProcessorFactory());
```

7. Cree `Scheduler` mediante `ConfigsBuilder` tal como se muestra en el ejemplo siguiente:

```
import java.util.UUID;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;

import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
```

```
...


DynamoDbAsyncClient dynamoClient =
 DynamoDbAsyncClient.builder().region(region).build();
CloudWatchAsyncClient cloudWatchClient =
 CloudWatchAsyncClient.builder().region(region).build();


DynamoDBStreamsPollingConfig pollingConfig = new
 DynamoDBStreamsPollingConfig(adapterClient);
pollingConfig.idleTimeBetweenReadsInMillis(idleTimeBetweenReadsInMillis);

// Use ConfigsBuilder to configure settings
RetrievalConfig retrievalConfig = configsBuilder.retrievalConfig();
retrievalConfig.retrievalSpecificConfig(pollingConfig);

CoordinatorConfig coordinatorConfig = configsBuilder.coordinatorConfig();
coordinatorConfig.clientVersionConfig(CoordinatorConfig.ClientVersionConfig.CLIENT_VERSION_

Scheduler scheduler = StreamsSchedulerFactory.createScheduler(
                configsBuilder.checkpointConfig(),
                coordinatorConfig,
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),
                retrievalConfig,
                adapterClient
        );
```

Important

La configuración `CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X` mantiene la compatibilidad entre el adaptador de Kinesis de DynamoDB Streams para KCL v3 y KCL v1, pero no entre KCL v2 y v3.

Paso 4: información general y recomendaciones sobre la configuración de KCL 3.x

Para obtener una descripción detallada de las configuraciones introducidas después de KCL 1.x que son relevantes en KCL 3.x, consulte las configuraciones de KCL y la configuración del cliente de migración de KCL.

> **Important**
>
> En lugar de crear directamente objetos de `checkpointConfig`, `coordinatorConfig`, `leaseManagementConfig`, `metricsConfig`, `processorConfig` y `retrievalConfig`, recomendamos utilizar `ConfigsBuilder` para establecer configuraciones en KCL 3.x y versiones posteriores y así evitar problemas de inicialización del programador. `ConfigsBuilder` proporciona una forma más flexible y sostenible de configurar la aplicación de KCL.

Configuraciones con valor predeterminado actualizado en KCL 3.x

`billingMode`

En la versión 1.x de KCL, el valor predeterminado de `billingMode` se establece en `PROVISIONED`. No obstante, con la versión 3.x de KCL, el valor predeterminado de `billingMode` es `PAY_PER_REQUEST` (modo bajo demanda). Le recomendamos que utilice el modo de capacidad bajo demanda para la tabla de arrendamiento a fin de ajustar automáticamente la capacidad en función del uso. Para obtener orientación sobre cómo utilizar la capacidad aprovisionada para las tablas de arrendamiento, consulte Best practices for the lease table with provisioned capacity mode.

`idleTimeBetweenReadsInMillis`

En la versión 1.x de KCL, el valor predeterminado de `idleTimeBetweenReadsInMillis` se establece en 1000 (o 1 segundo). La versión 3.x de KCL establece el valor predeterminado de `idleTimeBetweenReadsInMillis` en 1500 (o 1,5 segundos), pero el adaptador de Kinesis de Amazon DynamoDB Streams reemplaza el valor predeterminado por 1000 (o 1 segundo).

Nuevas configuraciones en KCL 3.x

`leaseAssignmentIntervalMillis`

Esta configuración define el intervalo de tiempo antes de que las particiones recién descubiertas comiencen a procesarse y se calcula como 1,5 × `leaseAssignmentIntervalMillis`. Si este ajuste no se configura explícitamente, el intervalo de tiempo se establece de forma predeterminada en 1,5 × `failoverTimeMillis`. El procesamiento de nuevas particiones implica examinar la tabla de arrendamiento y consultar un índice secundario global (GSI) en la tabla de arrendamiento. Al reducir `leaseAssignmentIntervalMillis`, aumenta la frecuencia de estas operaciones de análisis y consulta, lo que se traduce en mayores costos de DynamoDB. Recomendamos establecer este valor en 2000 (o 2 segundos) para minimizar el retraso en el procesamiento de nuevas particiones.

`shardConsumerDispatchPollIntervalMillis`

Esta configuración define el intervalo entre sondeos sucesivos por parte del consumidor de particiones para activar las transiciones de estado. En la versión 1.x de KCL, este comportamiento se controlaba mediante el parámetro `idleTimeInMillis`, que no se exponía como un ajuste configurable. Con la versión 3.x de KCL, recomendamos establecer esta configuración para que coincida con el valor utilizado para `idleTimeInMillis` en la configuración de la versión 1.x de KCL.

Paso 5: migración de KCL 2.x a KCL 3.x

Para garantizar una transición fluida y la compatibilidad con la última versión de la biblioteca de clientes de Kinesis (KCL), siga los pasos del 5 al 8 de las instrucciones de la guía de migración para actualizar de KCL 2.x a KCL 3.x.

Para ver los problemas habituales de solución de problemas de KCL 3.x, consulte Solución de problemas de las aplicaciones de consumidores de KCL.

Restauración de la versión de KCL anterior

En este tema se explica cómo restaurar la aplicación de consumidor a la versión de KCL anterior. El proceso de restauración consta de dos pasos:

1. Ejecución de la herramienta de migración de KCL.
2. Nueva implementación del código de la versión de KCL anterior.

Paso 1: ejecución de la herramienta de migración de KCL

Cuando necesite restaurar la versión anterior de KCL, debe ejecutar la herramienta de migración de KCL. La herramienta realiza dos tareas importantes:

- Elimina una tabla de metadatos llamada tabla de métricas de procesos de trabajo y el índice secundario global de la tabla de arrendamiento en DynamoDB. Estos artefactos los crea KCL 3.x, pero no son necesarios al restaurar la versión anterior.
- Hace que todos los procesos de trabajo se ejecuten en un modo compatible con KCL 1.x y comiencen a utilizar el algoritmo de equilibrio de carga utilizado en versiones anteriores de KCL. Si tiene problemas con el nuevo algoritmo de equilibrio de carga en KCL 3.x, esto mitigará el problema inmediatamente.

> **Important**
>
> La tabla de estados del coordinador en DynamoDB debe existir y no debe eliminarse durante el proceso de migración, restauración y avance.

> **Note**
>
> Es importante que todos los procesos de trabajo de la aplicación de consumo utilicen el mismo algoritmo de equilibrio de carga en un momento dado. La herramienta de migración de KCL se asegura de que todos los procesos de trabajo de la aplicación de consumo KCL 3.x cambien al modo compatible con KCL 1.x, de modo que todos los procesos de trabajo ejecuten el mismo algoritmo de equilibrio de carga durante la restauración de la aplicación a la versión anterior de KCL.

Puede descargar la herramienta de migración de KCL en el directorio de scripts del repositorio de GitHub de KCL. Ejecute el script desde un proceso de trabajo o host con los permisos adecuados para escribir en la tabla de estados del coordinador, la tabla de métricas de los procesos de trabajo y la tabla de arrendamiento. Asegúrese de que los permisos de IAM adecuados estén configurados para las aplicaciones de consumo de KCL. Ejecute el script solo una vez por aplicación de KCL mediante el comando especificado:

```
python3 ./KclMigrationTool.py --region region --mode rollback [--
application_name applicationName] [--lease_table_name leaseTableName]
 [--coordinator_state_table_name coordinatorStateTableName] [--
worker_metrics_table_name workerMetricsTableName]
```

Parameters

`--region`

Sustituya *region* por su Región de AWS.

`--application_name`

Este parámetro es necesario si utiliza nombres predeterminados para las tablas de metadatos de DynamoDB (tabla de arrendamiento, tabla de estados del coordinador y tabla de métricas de los proceso de trabajo). Si ha especificado nombres personalizados para estas tablas, puede omitir este parámetro. Reemplace *applicationName* por el nombre de la aplicación KCL real. La herramienta utiliza este nombre para derivar los nombres de tabla predeterminados si no se proporcionan nombres personalizados.

`--lease_table_name`

Este parámetro es necesario cuando ha establecido un nombre personalizado para la tabla de arrendamientos en la configuración de KCL. Si utiliza el nombre de tabla predeterminado, puede omitir este parámetro. Reemplace *leaseTableName* por el nombre de tabla personalizado que especificó para la tabla de arrendamiento.

`--coordinator_state_table_name`

Este parámetro es necesario cuando ha establecido un nombre personalizado para la tabla de estados de coordinador en la configuración de KCL. Si utiliza el nombre de tabla predeterminado, puede omitir este parámetro. Reemplace *coordinatorStateTableName* por el nombre de tabla personalizado que especificó para la tabla de estados de coordinador.

`--worker_metrics_table_name`

Este parámetro es necesario cuando ha establecido un nombre personalizado para la tabla de métricas de proceso de trabajo en la configuración de KCL. Si utiliza el nombre de tabla predeterminado, puede omitir este parámetro. Reemplace *workerMetricsTableName* por el nombre de tabla personalizado que especificó para la tabla de métricas de proceso de trabajo.

Paso 2: nueva implementación del código con la versión de KCL anterior

> **Important**
>
> Cualquier mención a la versión 2.x en la salida generada por la herramienta de migración de KCL debe interpretarse como una referencia a la versión 1.x de KCL. La ejecución del script no realiza una recuperación completa, solo cambia el algoritmo de equilibrio de carga por el utilizado en la versión 1.x de KCL.

Tras ejecutar la herramienta de migración de KCL para realizar una recuperación, verá uno de los siguientes mensajes:

Mensaje 1

> “Rollback completed. Your application was running 2x compatible functionality. Please rollback to your previous application binaries by deploying the code with your previous KCL version”.
>
> Acción requerida: esto significa que los procesos de trabajo se estaban ejecutando en el modo compatible con KCL 1.x. Vuelva a implementar el código con la versión de KCL anterior en los procesos de trabajo.

Mensaje 2

> “Rollback completed. Your KCL Application was running 3x functionality and will rollback to 2x compatible functionality. If you don't see mitigation after a short period of time, please rollback to your previous application binaries by deploying the code with your previous KCL version”.
>
> Acción requerida: esto significa que los procesos de trabajo se estaban ejecutando en modo KCL 3.x y la herramienta de migración de KCL ha cambiado todos los procesos de trabajo al modo compatible con KCL 1.x. Vuelva a implementar el código con la versión de KCL anterior en los procesos de trabajo.

Mensaje 3

> “Application was already rolled back. Any KCLv3 resources that could be deleted were cleaned up to avoid charges until the application can be rolled forward with migration”.
>
> Acción requerida: esto significa que los procesos de trabajo ya se han restaurado para ejecutarse en el modo compatible con KCL 1.x. Vuelva a implementar el código con la versión de KCL anterior en los procesos de trabajo.

Avance a KCL 3.x después de una restauración

En este tema se explica cómo avanzar la aplicación de consumidor a KCL 3.x después de una restauración. Cuando necesite avanzar, debe completar un proceso de dos pasos:

1. Ejecución de la herramienta de migración de KCL.
2. Implemente el código con KCL 3.x.

Paso 1: ejecución de la herramienta de migración de KCL

Ejecute la herramienta de migración de KCL con el siguiente comando para avanzar a KCL 3.x:

```
python3 ./KclMigrationTool.py --region region --mode rollforward [--
application_name applicationName] [--
coordinator_state_table_name coordinatorStateTableName]
```

Parameters

`--region`

Sustituya *region* por su Región de AWS.

`--application_name`

Este parámetro es obligatorio si utiliza nombres predeterminados para la tabla de estados de coordinador. Si ha especificado nombres personalizados para la tabla de estados de coordinador, puede omitir este parámetro. Reemplace *applicationName* por el nombre de la aplicación KCL real. La herramienta utiliza este nombre para derivar los nombres de tabla predeterminados si no se proporcionan nombres personalizados.

`--coordinator_state_table_name`

Este parámetro es necesario cuando ha establecido un nombre personalizado para la tabla de estados de coordinador en la configuración de KCL. Si utiliza el nombre de tabla predeterminado, puede omitir este parámetro. Reemplace *coordinatorStateTableName* por el nombre de tabla personalizado que especificó para la tabla de estados de coordinador.

Después de ejecutar la herramienta de migración en modo de avance, KCL crea los siguientes recursos de DynamoDB necesarios para KCL 3.x:

- Un índice secundario global en la tabla de arrendamientos

- Una tabla de métricas de proceso de trabajo

Paso 2: implementación del código con KCL 3.x

Después de ejecutar la herramienta de migración de KCL para una restauración, implemente el código con KCL 3.x en los procesos de trabajo. Para completar la migración, consulte Paso 8: complete la migración.

Tutorial: Adaptador Kinesis de DynamoDB Streams

En esta sección se explica paso a paso una aplicación Java en la que se utiliza Amazon Kinesis Client Library y Amazon DynamoDB Streams Kinesis Adapter. En la aplicación se muestra un ejemplo de replicación de datos, donde la actividad de escritura de una tabla se aplica a una segunda tabla, de tal forma que el contenido de ambas se mantiene sincronizado. Para obtener el código fuente, consulte Programa completo: DynamoDB Streams Kinesis Adapter.

El programa realiza lo siguiente:

1. Crea dos tablas de DynamoDB denominadas `KCL-Demo-src` y `KCL-Demo-dst`. En cada una de estas tablas se ha habilitado una secuencia.
2. Agrega, actualiza y elimina elementos para generar actividad de actualización en la tabla de origen. Esto hace que se escriban datos en la secuencia de la tabla.
3. Lee los registros en la transmisión, los reconstruye como solicitudes de DynamoDB y aplica las solicitudes a la tabla de destino.
4. Examina las tablas de origen y destino para comprobar que sus contenidos sean idénticos.
5. Efectúa una limpieza eliminando las tablas.

Estos pasos se describen en las siguientes secciones y la aplicación completa se muestra al final del tutorial.

Temas

- Paso 1: crear tablas de DynamoDB
- Paso 2: generar actividad de actualización en la tabla de origen
- Paso 3: procesar la secuencia
- Paso 4: comprobar que el contenido de ambas tablas es idéntico
- Paso 5: Eliminar

- Programa completo: DynamoDB Streams Kinesis Adapter

Paso 1: crear tablas de DynamoDB

El primer paso consiste en crear dos tablas de DynamoDB, una de origen y una de destino. El `StreamViewType` de la secuencia de la tabla de origen es `NEW_IMAGE`. Esto significa que cada vez que se modifica un elemento en esta tabla, su imagen de "después" se escribe en la secuencia. De esta forma, se realiza un seguimiento en la secuencia de todas las actividades de escritura en la tabla.

En el siguiente ejemplo se muestra el código utilizado para crear las dos tablas.

```
java.util.List<AttributeDefinition> attributeDefinitions = new
 ArrayList<AttributeDefinition>();
attributeDefinitions.add(new
 AttributeDefinition().withAttributeName("Id").withAttributeType("N"));

java.util.List<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
keySchema.add(new
 KeySchemaElement().withAttributeName("Id").withKeyType(KeyType.HASH)); // Partition

  // key

ProvisionedThroughput provisionedThroughput = new
 ProvisionedThroughput().withReadCapacityUnits(2L)
    .withWriteCapacityUnits(2L);

StreamSpecification streamSpecification = new StreamSpecification();
streamSpecification.setStreamEnabled(true);
streamSpecification.setStreamViewType(StreamViewType.NEW_IMAGE);
CreateTableRequest createTableRequest = new
 CreateTableRequest().withTableName(tableName)
    .withAttributeDefinitions(attributeDefinitions).withKeySchema(keySchema)

 .withProvisionedThroughput(provisionedThroughput).withStreamSpecification(streamSpecification)
```

Paso 2: generar actividad de actualización en la tabla de origen

El siguiente paso consiste en generar actividad de escritura en la tabla de origen. Mientras tiene lugar esta actividad, la secuencia de la tabla de origen también se actualiza casi en tiempo real.

En la aplicación se define una clase auxiliar con métodos que llaman a las operaciones de API `PutItem`, `UpdateItem` y `DeleteItem` para escribir los datos. En el siguiente ejemplo se muestra cómo se utilizan estos métodos.

```
StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName, "101", "test1");
StreamsAdapterDemoHelper.updateItem(dynamoDBClient, tableName, "101", "test2");
StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName, "101");
StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName, "102", "demo3");
StreamsAdapterDemoHelper.updateItem(dynamoDBClient, tableName, "102", "demo4");
StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName, "102");
```

Paso 3: procesar la secuencia

Ahora, el programa comienza a procesar la secuencia. DynamoDB Streams Kinesis Adapter actúa como una capa transparente entre la KCL y el punto de enlace de DynamoDB Streams, para que el código pueda utilizar plenamente la KCL, en lugar de tener que realizar llamadas a DynamoDB Streams de bajo nivel. En el programa se realizan las siguientes tareas:

- Se define una clase de procesador de registros, `StreamsRecordProcessor`, con métodos que cumplen con la definición de interfaz de KCL: `initialize`, `processRecords` y `shutdown`. El método `processRecords` contiene la lógica necesaria para leer la secuencia de la tabla de origen y escribir en la tabla de destino.
- Define un generador de clases para la clase de procesador de registros (`StreamsRecordProcessorFactory`). Esto es necesario para los programas Java que utilizan la KCL.
- Crea una nueva instancia del proceso de trabajo `Worker` de la KCL, asociado con el generador de clases.
- Cierra el proceso de trabajo `Worker` cuando ha finalizado de procesar registros.

Para obtener más información sobre la definición de la interfaz de KCL, consulte Desarrollo de consumidores mediante la biblioteca Kinesis Client Library en la Guía de desarrolladores de Amazon Kinesis Data Streams.

En el siguiente ejemplo se muestra el bucle principal de `StreamsRecordProcessor`. La instrucción `case` determina qué acción se debe llevar a cabo, según el valor de `OperationType` que aparece en el registro de secuencia.

```
for (Record record : records) {
    String data = new String(record.getData().array(), Charset.forName("UTF-8"));
    System.out.println(data);
    if (record instanceof RecordAdapter) {
                software.amazon.dynamodb.model.Record streamRecord = ((RecordAdapter)
 record)
                    .getInternalObject();

                switch (streamRecord.getEventName()) {
                    case "INSERT":
                    case "MODIFY":
                        StreamsAdapterDemoHelper.putItem(dynamoDBClient, tableName,
                            streamRecord.getDynamodb().getNewImage());
                        break;
                    case "REMOVE":
                        StreamsAdapterDemoHelper.deleteItem(dynamoDBClient, tableName,
                            streamRecord.getDynamodb().getKeys().get("Id").getN());
                }
    }
    checkpointCounter += 1;
    if (checkpointCounter % 10 == 0) {
        try {
            checkpointer.checkpoint();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Paso 4: comprobar que el contenido de ambas tablas es idéntico

En este punto, el contenido de las tablas de origen y destino está sincronizado. La aplicación emite solicitudes `Scan` en las dos tablas para comprobar que su contenido sea realmente idéntico.

La clase `DemoHelper` contiene un método `ScanTable` que llama a la API de bajo nivel `Scan`. El siguiente ejemplo le muestra cómo se usa.

```
if (StreamsAdapterDemoHelper.scanTable(dynamoDBClient, srcTable).getItems()
    .equals(StreamsAdapterDemoHelper.scanTable(dynamoDBClient, destTable).getItems()))
 {
```

```
    System.out.println("Scan result is equal.");
}
else {
    System.out.println("Tables are different!");
}
```

Paso 5: Eliminar

La demostración ha finalizado. Por consiguiente, la aplicación elimina las tablas de origen y destino. Consulte el siguiente ejemplo de código. Incluso después de que las tablas se hayan eliminado, sus secuencias permanecerán disponibles durante un máximo de 24 horas; transcurrido este periodo se eliminan automáticamente.

```
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(srcTable));
dynamoDBClient.deleteTable(new DeleteTableRequest().withTableName(destTable));
```

Programa completo: DynamoDB Streams Kinesis Adapter

A continuación encontrará el programa de Java completo que lleva a cabo las tareas descritas en Tutorial: Adaptador Kinesis de DynamoDB Streams. Cuando lo ejecute, debería ver un resultado similar al siguiente.

```
Creating table KCL-Demo-src
Creating table KCL-Demo-dest
Table is active.
Creating worker for stream: arn:aws:dynamodb:us-west-2:111122223333:table/KCL-Demo-src/
stream/2015-05-19T22:48:56.601
Starting worker...
Scan result is equal.
Done.
```

> **Important**
>
> Para ejecutar este programa, utilice políticas con el fin de asegurarse de que la aplicación cliente tenga acceso a DynamoDB y a Amazon CloudWatch. Para obtener más información, consulte Políticas basadas en identidad de DynamoDB.

El código fuente consta de cuatro archivos `.java`:

- StreamsAdapterDemo.java
- StreamsRecordProcessor.java
- StreamsRecordProcessorFactory.java
- StreamsAdapterDemoHelper.java

StreamsAdapterDemo.java

```
package com.amazonaws.codesamples;

import
 com.amazonaws.services.dynamodbv2.streamsadapter.AmazonDynamoDBStreamsAdapterClient;
import com.amazonaws.services.dynamodbv2.streamsadapter.StreamsSchedulerFactory;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.polling.DynamoDBStreamsPollingConfig;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.InitialPositionInStream;
import software.amazon.kinesis.common.InitialPositionInStreamExtended;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;
import software.amazon.kinesis.processor.StreamTracker;
import software.amazon.kinesis.retrieval.RetrievalConfig;

public class StreamsAdapterDemo {

    private static DynamoDbAsyncClient dynamoDbAsyncClient;
    private static CloudWatchAsyncClient cloudWatchAsyncClient;
    private static AmazonDynamoDBStreamsAdapterClient
 amazonDynamoDbStreamsAdapterClient;

    private static String tablePrefix = "KCL-Demo";
    private static String streamArn;

    private static Region region = Region.US_EAST_1;
```

```
   private static AwsCredentialsProvider credentialsProvider =
DefaultCredentialsProvider.create();

   public static void main( String[] args ) throws Exception {
       System.out.println("Starting demo...");
       dynamoDbAsyncClient = DynamoDbAsyncClient.builder()
               .credentialsProvider(credentialsProvider)
               .region(region)
               .build();
       cloudWatchAsyncClient = CloudWatchAsyncClient.builder()
               .credentialsProvider(credentialsProvider)
               .region(region)
               .build();
       amazonDynamoDbStreamsAdapterClient = new
AmazonDynamoDBStreamsAdapterClient(credentialsProvider, region);

       String srcTable = tablePrefix + "-src";
       String destTable = tablePrefix + "-dest";

       setUpTables();

       StreamTracker streamTracker =
StreamsSchedulerFactory.createSingleStreamTracker(streamArn,

InitialPositionInStreamExtended.newInitialPosition(InitialPositionInStream.TRIM_HORIZON));

       ShardRecordProcessorFactory shardRecordProcessorFactory =
               new StreamsAdapterDemoProcessorFactory(dynamoDbAsyncClient, destTable);

       ConfigsBuilder configsBuilder = new ConfigsBuilder(
               streamTracker,
               "streams-adapter-demo",
               amazonDynamoDbStreamsAdapterClient,
               dynamoDbAsyncClient,
               cloudWatchAsyncClient,
               "streams-demo-worker",
               shardRecordProcessorFactory
       );

       DynamoDBStreamsPollingConfig pollingConfig = new
DynamoDBStreamsPollingConfig(amazonDynamoDbStreamsAdapterClient);
       RetrievalConfig retrievalConfig = configsBuilder.retrievalConfig();
       retrievalConfig.retrievalSpecificConfig(pollingConfig);
```

```
        System.out.println("Creating scheduler for stream " + streamArn);
        Scheduler scheduler = StreamsSchedulerFactory.createScheduler(
                configsBuilder.checkpointConfig(),
                configsBuilder.coordinatorConfig(),
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
                configsBuilder.processorConfig(),
                retrievalConfig,
                amazonDynamoDbStreamsAdapterClient
        );

        System.out.println("Starting scheduler...");
        Thread t = new Thread(scheduler);
        t.start();

        Thread.sleep(250000);

        System.out.println("Stopping scheduler...");
        scheduler.shutdown();
        t.join();

        if (StreamsAdapterDemoHelper.scanTable(dynamoDbAsyncClient, srcTable).items()
                .equals(StreamsAdapterDemoHelper.scanTable(dynamoDbAsyncClient,
destTable).items())) {
            System.out.println("Scan result is equal.");
        } else {
            System.out.println("Tables are different!");
        }

        System.out.println("Done.");
        cleanupAndExit(0);
    }

    private static void setUpTables() {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";
        streamArn = StreamsAdapterDemoHelper.createTable(dynamoDbAsyncClient,
srcTable);
        StreamsAdapterDemoHelper.createTable(dynamoDbAsyncClient, destTable);

        awaitTableCreation(srcTable);

        performOps(srcTable);
```

```
    }

    private static void awaitTableCreation(String tableName) {
        Integer retries = 0;
        Boolean created = false;
        while (!created && retries < 100) {
            DescribeTableResponse result =
StreamsAdapterDemoHelper.describeTable(dynamoDbAsyncClient, tableName);
            created = result.table().tableStatusAsString().equals("ACTIVE");
            if (created) {
                System.out.println("Table is active.");
                return;
            } else {
                retries++;
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    // do nothing
                }
            }
        }
        System.out.println("Timeout after table creation. Exiting...");
        cleanupAndExit(1);
    }

    private static void performOps(String tableName) {
        StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName, "101",
"test1");
        StreamsAdapterDemoHelper.updateItem(dynamoDbAsyncClient, tableName, "101",
"test2");
        StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName, "101");
        StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName, "102",
"demo3");
        StreamsAdapterDemoHelper.updateItem(dynamoDbAsyncClient, tableName, "102",
"demo4");
        StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName, "102");
    }

    private static void cleanupAndExit(Integer returnValue) {
        String srcTable = tablePrefix + "-src";
        String destTable = tablePrefix + "-dest";

dynamoDbAsyncClient.deleteTable(DeleteTableRequest.builder().tableName(srcTable).build());
```

```
 dynamoDbAsyncClient.deleteTable(DeleteTableRequest.builder().tableName(destTable).build());
        System.exit(returnValue);
    }
}
```

StreamsRecordProcessor.java

```
package com.amazonaws.codesamples;

import
 com.amazonaws.services.dynamodbv2.streamsadapter.adapter.DynamoDBStreamsClientRecord;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.model.DynamoDBStreamsProcessRecordsInput;
import
 com.amazonaws.services.dynamodbv2.streamsadapter.processor.DynamoDBStreamsShardRecordProcessor
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.Record;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;

import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;

public class StreamsRecordProcessor implements DynamoDBStreamsShardRecordProcessor {

    private Integer checkpointCounter;

    private final DynamoDbAsyncClient dynamoDbAsyncClient;
    private final String tableName;

    public StreamsRecordProcessor(DynamoDbAsyncClient dynamoDbAsyncClient, String
 tableName) {
        this.dynamoDbAsyncClient = dynamoDbAsyncClient;
        this.tableName = tableName;
    }

    @Override
    public void initialize(InitializationInput initializationInput) {
```

```
        this.checkpointCounter = 0;
    }

    @Override
    public void processRecords(DynamoDBStreamsProcessRecordsInput
dynamoDBStreamsProcessRecordsInput) {
        for (DynamoDBStreamsClientRecord record:
dynamoDBStreamsProcessRecordsInput.records()) {
            String data = new String(record.data().array(), StandardCharsets.UTF_8);
            System.out.println(data);
            Record streamRecord = record.getRecord();

            switch (streamRecord.eventName()) {
                case INSERT:
                case MODIFY:
                    StreamsAdapterDemoHelper.putItem(dynamoDbAsyncClient, tableName,
                            streamRecord.dynamodb().newImage());
                case REMOVE:
                    StreamsAdapterDemoHelper.deleteItem(dynamoDbAsyncClient, tableName,
                            streamRecord.dynamodb().keys().get("Id").n());
            }
            checkpointCounter += 1;
            if (checkpointCounter % 10 == 0) {
                try {
                    dynamoDBStreamsProcessRecordsInput.checkpointer().checkpoint();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {
        System.out.println("Lease Lost");
    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            e.printStackTrace();
        }
```

```
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            e.printStackTrace();
        }
    }
}
```

StreamsRecordProcessorFactory.java

```
package com.amazonaws.codesamples;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class StreamsAdapterDemoProcessorFactory implements ShardRecordProcessorFactory
 {
    private final String tableName;
    private final DynamoDbAsyncClient dynamoDbAsyncClient;

    public StreamsAdapterDemoProcessorFactory(DynamoDbAsyncClient asyncClient, String
 tableName) {
        this.tableName = tableName;
        this.dynamoDbAsyncClient = asyncClient;
    }

    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new StreamsRecordProcessor(dynamoDbAsyncClient, tableName);
    }
}
```

StreamsAdapterDemoHelper.java

```
package com.amazonaws.codesamples;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
```

```
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.OnDemandThroughput;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.ResourceInUseException;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import software.amazon.awssdk.services.dynamodb.model.StreamSpecification;
import software.amazon.awssdk.services.dynamodb.model.StreamViewType;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class StreamsAdapterDemoHelper {

    /**
     * @return StreamArn
     */
    public static String createTable(DynamoDbAsyncClient client, String tableName) {
        List<AttributeDefinition> attributeDefinitions = new ArrayList<>();
        attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName("Id")
                .attributeType("N")
                .build());

        List<KeySchemaElement> keySchema = new ArrayList<>();
        keySchema.add(KeySchemaElement.builder()
                .attributeName("Id")
                .keyType(KeyType.HASH) // Partition key
                .build());

        StreamSpecification streamSpecification = StreamSpecification.builder()
                .streamEnabled(true)
                .streamViewType(StreamViewType.NEW_IMAGE)
```

```
                .build();

        CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .tableName(tableName)
                .attributeDefinitions(attributeDefinitions)
                .keySchema(keySchema)
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .streamSpecification(streamSpecification)
                .build();

        try {
            System.out.println("Creating table " + tableName);
            CreateTableResponse result = client.createTable(createTableRequest).join();
            return result.tableDescription().latestStreamArn();
        } catch (Exception e) {
            if (e.getCause() instanceof ResourceInUseException) {
                System.out.println("Table already exists.");
                return describeTable(client, tableName).table().latestStreamArn();
            }
            throw e;
        }
    }

    public static DescribeTableResponse describeTable(DynamoDbAsyncClient client,
String tableName) {
        return client.describeTable(DescribeTableRequest.builder()
                        .tableName(tableName)
                        .build())
                .join();
    }

    public static ScanResponse scanTable(DynamoDbAsyncClient dynamoDbClient, String
tableName) {
        return dynamoDbClient.scan(ScanRequest.builder()
                        .tableName(tableName)
                        .build())
                .join();
    }

    public static void putItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
String id, String val) {
        Map<String, AttributeValue> item = new HashMap<>();
        item.put("Id", AttributeValue.builder().n(id).build());
        item.put("attribute-1", AttributeValue.builder().s(val).build());
```

```
        putItem(dynamoDbClient, tableName, item);
    }

    public static void putItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
                               Map<String, AttributeValue> items) {
        PutItemRequest putItemRequest = PutItemRequest.builder()
                .tableName(tableName)
                .item(items)
                .build();
        dynamoDbClient.putItem(putItemRequest).join();
    }

    public static void updateItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
String id, String val) {
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("Id", AttributeValue.builder().n(id).build());

        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#attr2", "attribute-2");

        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":val", AttributeValue.builder().s(val).build());

        UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #attr2 = :val")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();

        dynamoDbClient.updateItem(updateItemRequest).join();
    }

    public static void deleteItem(DynamoDbAsyncClient dynamoDbClient, String tableName,
String id) {
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("Id", AttributeValue.builder().n(id).build());

        DeleteItemRequest deleteItemRequest = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .build();
```

```
        dynamoDbClient.deleteItem(deleteItemRequest).join();
    }
}
```

API de bajo nivel de DynamoDB Streams: ejemplo en Java

> **Note**
>
> El código que se presenta en esta página no es exhaustivo y no contempla todos los escenarios de consumo de Amazon DynamoDB Streams. La manera recomendada de consumir registros de transmisión de DynamoDB consiste en usar Amazon Kinesis Adapter con la Kinesis Client Library (KCL), como se describe en Uso del adaptador Kinesis de DynamoDB Streams para procesar registros de transmisión.

Esta sección contiene un programa de Java que muestra el funcionamiento de DynamoDB Streams. El programa realiza lo siguiente:

1. Crea una tabla de DynamoDB con una transmisión habilitada.
2. Describe los ajustes de secuencia de esta tabla.
3. Modifica los datos de la tabla.
4. Describe los fragmentos de la secuencia.
5. Lee los registros de secuencia de los fragmentos.
6. Busca las particiones secundarias y continúa leyendo los registros.
7. Elimina recursos.

Al ejecutar el programa, verá un resultado parecido al siguiente.

```
Testing Streams Demo
Creating an Amazon DynamoDB table TestTableForStreams with a simple primary key: Id
Waiting for TestTableForStreams to be created...
Current stream ARN for TestTableForStreams: arn:aws:dynamodb:us-
east-2:123456789012:table/TestTableForStreams/stream/2018-03-20T16:49:55.208
Stream enabled: true
Update view type: NEW_AND_OLD_IMAGES

Performing write activities on TestTableForStreams
Processing item 1 of 100
```

```
Processing item 2 of 100
Processing item 3 of 100
...
Processing item 100 of 100
Shard: {ShardId: shardId-1234567890-...,SequenceNumberRange: {StartingSequenceNumber:
 100002572486797508907,},}
    Shard iterator: EjYFEkX2a26eVTWe...
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, NewImage={Message=AttributeValue(S=New Item!),
 Id=AttributeValue(S=4)}, SequenceNumber=200000158404754583390 9, SizeBytes=22,
 StreamViewType=NEW_AND_OLD_IMAGES)
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, NewImage={Message=AttributeValue(S=This is an updated
 item), Id=AttributeValue(S=4)}, OldImage={Message=AttributeValue(S=New Item!),
 Id=AttributeValue(S=4)}, SequenceNumber=210000360486976789270 1, SizeBytes=55,
 StreamViewType=NEW_AND_OLD_IMAGES)
        StreamRecord(ApproximateCreationDateTime=2025-04-09T13:11:58Z,
 Keys={Id=AttributeValue(S=4)}, OldImage={Message=AttributeValue(S=This is an updated
 item), Id=AttributeValue(S=4)}, SequenceNumber=220000109977111289843 4, SizeBytes=36,
 StreamViewType=NEW_AND_OLD_IMAGES)
...
Deleting the table...
Table StreamsDemoTable deleted.
Demo complete
```

Example Ejemplo

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
```

```
import software.amazon.awssdk.services.dynamodb.model.DescribeStreamRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeStreamResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetRecordsRequest;
import software.amazon.awssdk.services.dynamodb.model.GetRecordsResponse;
import software.amazon.awssdk.services.dynamodb.model.GetShardIteratorRequest;
import software.amazon.awssdk.services.dynamodb.model.GetShardIteratorResponse;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.Record;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.Shard;
import software.amazon.awssdk.services.dynamodb.model.ShardFilter;
import software.amazon.awssdk.services.dynamodb.model.ShardFilterType;
import software.amazon.awssdk.services.dynamodb.model.ShardIteratorType;
import software.amazon.awssdk.services.dynamodb.model.StreamSpecification;
import software.amazon.awssdk.services.dynamodb.model.TableDescription;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.streams.DynamoDbStreamsClient;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

public class StreamsLowLevelDemo {


    public static void main(String[] args) {
        final String usage = "Testing Streams Demo";
        try {
            System.out.println(usage);

            String tableName = "StreamsDemoTable";
            String key = "Id";
            System.out.println("Creating an Amazon DynamoDB table " + tableName + "
 with a simple primary key: " + key);
            Region region = Region.US_WEST_2;
            DynamoDbClient ddb = DynamoDbClient.builder()
                    .region(region)
                    .build();

            DynamoDbStreamsClient ddbStreams = DynamoDbStreamsClient.builder()
                    .region(region)
                    .build();
```

```
            DescribeTableRequest describeTableRequest = DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();
            TableDescription tableDescription = null;
            try{
                tableDescription = ddb.describeTable(describeTableRequest).table();
            }catch (Exception e){
                System.out.println("Table " + tableName + " does not exist.");
                tableDescription = createTable(ddb, tableName, key);
            }

            // Print the stream settings for the table
            String streamArn = tableDescription.latestStreamArn();

            StreamSpecification streamSpec = tableDescription.streamSpecification();
            System.out.println("Current stream ARN for " + tableDescription.tableName()
+ ": " +
                   streamArn);
            System.out.println("Stream enabled: " + streamSpec.streamEnabled());
            System.out.println("Update view type: " + streamSpec.streamViewType());
            System.out.println();
            // Generate write activity in the table
            System.out.println("Performing write activities on " + tableName);
            int maxItemCount = 100;
            for (Integer i = 1; i <= maxItemCount; i++) {
                System.out.println("Processing item " + i + " of " + maxItemCount);
                // Write a new item
                putItemInTable(key, i, tableName, ddb);
                // Update the item
                updateItemInTable(key, i, tableName, ddb);
                // Delete the item
                deleteDynamoDBItem(key, i, tableName, ddb);
            }

            // Process Stream
            processStream(streamArn, maxItemCount, ddb, ddbStreams, tableName);

            // Delete the table
            System.out.println("Deleting the table...");
            DeleteTableRequest deleteTableRequest = DeleteTableRequest.builder()
                    .tableName(tableName)
                    .build();
            ddb.deleteTable(deleteTableRequest);
            System.out.println("Table " + tableName + " deleted.");
```

```
            System.out.println("Demo complete");
            ddb.close();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    private static void processStream(String streamArn, int maxItemCount,
DynamoDbClient ddb, DynamoDbStreamsClient ddbStreams, String tableName) {
        // Get all the shard IDs from the stream. Note that DescribeStream returns
        // the shard IDs one page at a time.
        String lastEvaluatedShardId = null;
        do {
            DescribeStreamRequest describeStreamRequest =
DescribeStreamRequest.builder()
                    .streamArn(streamArn)
                    .exclusiveStartShardId(lastEvaluatedShardId).build();
            DescribeStreamResponse describeStreamResponse =
ddbStreams.describeStream(describeStreamRequest);

            List<Shard> shards = describeStreamResponse.streamDescription().shards();

            // Process each shard on this page

            fetchShardsAndReadRecords(streamArn, maxItemCount, ddbStreams, shards);

            // If LastEvaluatedShardId is set, then there is
            // at least one more page of shard IDs to retrieve
            lastEvaluatedShardId =
describeStreamResponse.streamDescription().lastEvaluatedShardId();

        } while (lastEvaluatedShardId != null);

    }

    private static void fetchShardsAndReadRecords(String streamArn, int maxItemCount,
DynamoDbStreamsClient ddbStreams, List<Shard> shards) {
        for (Shard shard : shards) {
            String shardId = shard.shardId();
            System.out.println("Shard: " + shard);

            // Get an iterator for the current shard
            GetShardIteratorRequest shardIteratorRequest =
GetShardIteratorRequest.builder()
```

```
                    .streamArn(streamArn).shardId(shardId)
                    .shardIteratorType(ShardIteratorType.TRIM_HORIZON).build();

            GetShardIteratorResponse getShardIteratorResult =
ddbStreams.getShardIterator(shardIteratorRequest);

            String currentShardIter = getShardIteratorResult.shardIterator();

            // Shard iterator is not null until the Shard is sealed (marked as
READ_ONLY).
            // To prevent running the loop until the Shard is sealed, we process only
the
            // items that were written into DynamoDB and then exit.
            int processedRecordCount = 0;
            while (currentShardIter != null && processedRecordCount < maxItemCount) {
                // Use the shard iterator to read the stream records
                GetRecordsRequest getRecordsRequest = GetRecordsRequest.builder()
                        .shardIterator(currentShardIter).build();
                GetRecordsResponse getRecordsResult =
ddbStreams.getRecords(getRecordsRequest);
                List<Record> records = getRecordsResult.records();
                for (Record record : records) {
                    System.out.println("        " + record.dynamodb());
                }
                processedRecordCount += records.size();
                currentShardIter = getRecordsResult.nextShardIterator();
            }
            if (currentShardIter == null){
                System.out.println("Shard has been fully processed. Shard iterator is
null.");
                System.out.println("Fetch the child shard to continue processing
instead of bulk fetching all shards");
                DescribeStreamRequest describeStreamRequestForChildShards =
DescribeStreamRequest.builder()
                        .streamArn(streamArn)
                        .shardFilter(ShardFilter.builder()
                                .type(ShardFilterType.CHILD_SHARDS)
                                .shardId(shardId).build())
                        .build();
                DescribeStreamResponse describeStreamResponseChildShards =
ddbStreams.describeStream(describeStreamRequestForChildShards);
                fetchShardsAndReadRecords(streamArn, maxItemCount, ddbStreams,
describeStreamResponseChildShards.streamDescription().shards());
            }
```

```
        }
    }

    private static void putItemInTable(String key, Integer i, String tableName,
DynamoDbClient ddb) {
        Map<String, AttributeValue> item = new HashMap<>();
        item.put(key, AttributeValue.builder()
                .s(i.toString())
                .build());
        item.put("Message", AttributeValue.builder()
                .s("New Item!")
                .build());
        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(item)
                .build();
        ddb.putItem(request);
    }

    private static void updateItemInTable(String key, Integer i, String tableName,
DynamoDbClient ddb) {

        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put(key, AttributeValue.builder()
                .s(i.toString())
                .build());


        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put("Message", AttributeValueUpdate.builder()
                .value(AttributeValue.builder().s("This is an updated item").build())
                .action(AttributeAction.PUT)
                .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(itemKey)
                .attributeUpdates(updatedValues)
                .build();
        ddb.updateItem(request);
    }

    public static void deleteDynamoDBItem(String key, Integer i, String tableName,
DynamoDbClient ddb) {
```

```
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(i.toString())
                .build());

        DeleteItemRequest deleteReq = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(keyToGet)
                .build();
        ddb.deleteItem(deleteReq);
    }

    public static TableDescription createTable(DynamoDbClient ddb, String tableName,
String key) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        StreamSpecification streamSpecification = StreamSpecification.builder()
                .streamEnabled(true)
                .streamViewType("NEW_AND_OLD_IMAGES")
                .build();
        CreateTableRequest request = CreateTableRequest.builder()
                .attributeDefinitions(AttributeDefinition.builder()
                        .attributeName(key)
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(KeySchemaElement.builder()
                        .attributeName(key)
                        .keyType(KeyType.HASH)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
                .tableName(tableName)
                .streamSpecification(streamSpecification)
                .build();

        TableDescription newTable;
        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build();

            System.out.println("Waiting for " + tableName + " to be created...");

            // Wait until the Amazon DynamoDB table is created.
```

```
            WaiterResponse<DescribeTableResponse> waiterResponse =
 dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            newTable = response.tableDescription();
            return newTable;

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return null;
    }



}
```

DynamoDB Streams y disparadores de AWS Lambda

Amazon DynamoDB se integra con AWS Lambda para que pueda crear desencadenadores, a saber, fragmentos de código que responden automáticamente a los eventos de DynamoDB Streams. Con los disparadores, puede crear aplicaciones que reaccionan ante las modificaciones de datos en las tablas de DynamoDB.

Temas

- Tutorial n.º 1: Uso de filtros para procesar todos los eventos con Amazon DynamoDB y AWS Lambda mediante la AWS CLI
- Tutorial n.º 2: uso de filtros para procesar algunos eventos con DynamoDB y Lambda
- Prácticas recomendadas para usar DynamoDB Streams con Lambda

Si habilita DynamoDB Streams en una tabla, puede asociar el nombre de recurso de Amazon (ARN) de la transmisión con una función de AWS Lambda que haya escrito. Todas las acciones de mutación a esa tabla de DynamoDB pueden capturarse como un elemento en el flujo. Por ejemplo, puede establecer un desencadenador para que, cuando se modifique un elemento de una tabla, aparezca inmediatamente un nuevo registro en el flujo de esa tabla.

> **Note**
>
> Si suscribe más de dos funciones de Lambda a un solo flujo de DynamoDB, podría aplicarse una limitación en la lectura.

El servicio AWS Lambda sondea el flujo en busca de nuevos registros cuatro veces por segundo. Cuando hay nuevos registros de flujo disponibles, se invoca su función Lambda de forma sincrónica. Puede suscribir hasta dos funciones Lambda al mismo flujo de DynamoDB. Si suscribe más de dos funciones de Lambda al mismo flujo de DynamoDB, podría aplicarse una limitación en la lectura.

La función Lambda puede enviar una notificación, iniciar un flujo de trabajo o realizar otras muchas acciones que le especifique. Puede escribir una función Lambda para que simplemente copie cada registro de flujo a un almacenamiento persistente, como la puerta de enlace de archivo de Amazon S3 (Amazon S3), y crear un registro de auditoría permanente de la actividad de escritura en su tabla. O bien suponga que tiene una aplicación de juegos para móviles que escribe en una tabla `GameScores`. Cada vez que se actualiza el atributo `TopScore` de la tabla `GameScores`, se escribe el registro correspondiente en la secuencia de la tabla. Este evento, a su vez, puede activar una función Lambda que publique un mensaje de felicitación en una red social. Esta función también podría escribirse para ignorar cualquier registro de flujo que no sea una actualización de `GameScores` o que no modifique el atributo `TopScore`.

Sin embargo, si la función devuelve un error, Lambda vuelve a intentar ejecutar el lote hasta que se procese correctamente o los datos caduquen. También puede configurar Lambda para que lo reintente con un lote más pequeño, limitar el número de reintentos, descartar los registros cuando sean demasiado antiguos y otras opciones.

Como práctica recomendada de rendimiento, la función Lambda debe ser de corta duración. Para evitar introducir retrasos innecesarios en el procesamiento, tampoco debe ejecutar una lógica compleja. Para un flujo de alta velocidad en concreto, es mejor desencadenar un flujo de trabajo asíncrono de funciones de posprocesamiento que funciones Lambda sincrónicas de larga duración.

No puede utilizar el mismo desencadenador Lambda en diferentes cuentas de AWS. Tanto la tabla de DynamoDB como las funciones de Lambda deben pertenecer a la misma cuenta de AWS.

Para obtener más información sobre AWS Lambda, consulte la Guía para desarrolladores de AWS Lambda.

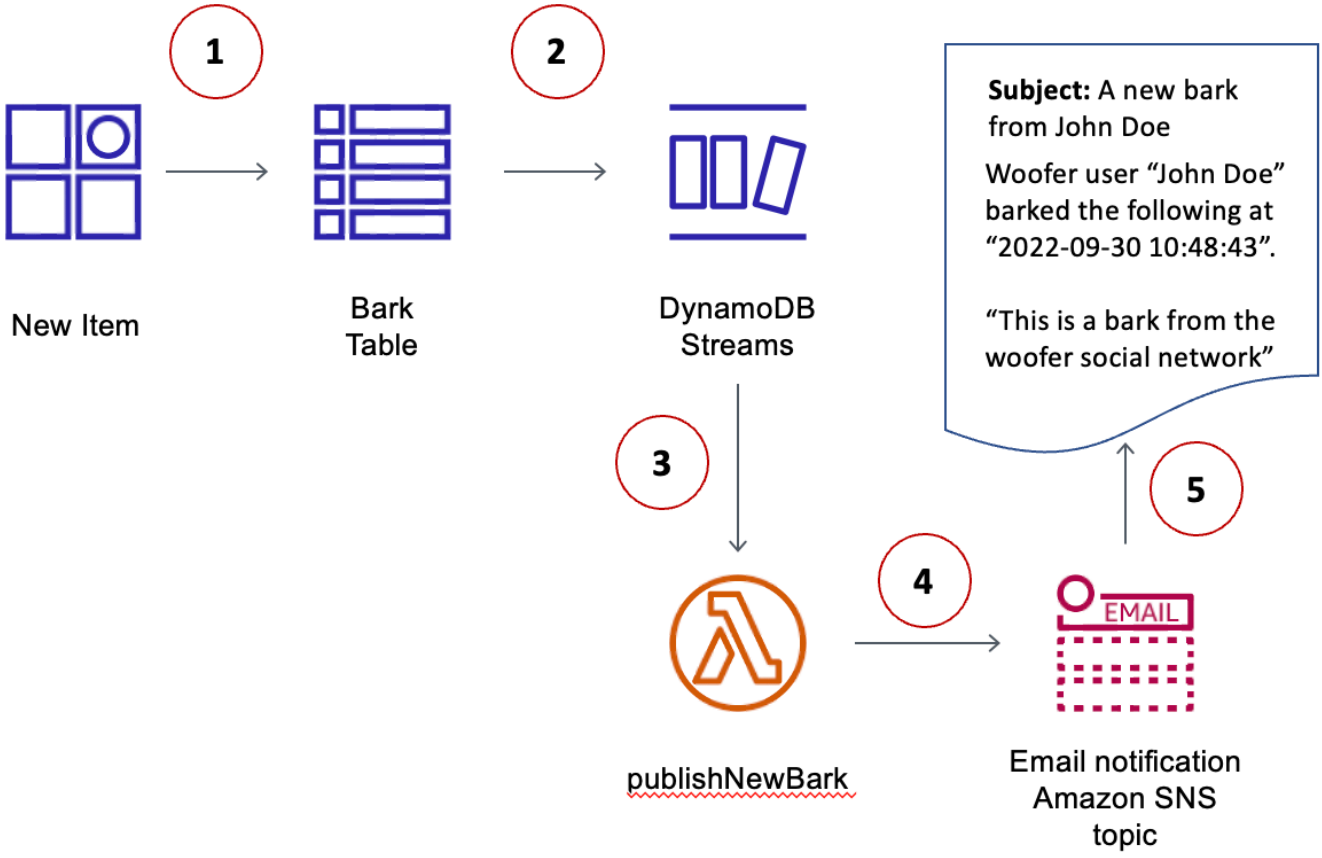
Tutorial n.º 1: Uso de filtros para procesar todos los eventos con Amazon DynamoDB y AWS Lambda mediante la AWS CLI

En este tutorial, creará un desencadenador AWS Lambda para procesar una transmisión de una tabla de DynamoDB.

Temas

- Paso 1: crear una tabla de DynamoDB con un flujo habilitado
- Paso 2: crear un rol de ejecución para Lambda
- Paso 3: crear un tema de Amazon SNS
- Paso 4: crear y probar una función Lambda
- Paso 5: crear y probar un disparador

El escenario en que tiene lugar este tutorial es Woofer, una red social sencilla. Los usuarios de Woofer se comunican utilizando ladridos (mensajes de texto breves) que se envían a otros usuarios de Woofer. En el siguiente diagrama se muestran los componentes y el flujo de trabajo de esta aplicación.



1. Un usuario escribe un elemento en una tabla de DynamoDB (`BarkTable`). Cada elemento de la tabla representa un ladrido.
2. Se escribe un nuevo registro de secuencia para reflejar que se ha agregado un elemento nuevo a `BarkTable`.
3. El nuevo registro de secuencia activa una función de AWS Lambda (`publishNewBark`).
4. Si el registro de transmisión indica que se ha agregado un nuevo elemento a `BarkTable`, la función de Lambda lee los datos del registro de transmisión y publica un mensaje en un tema de Amazon Simple Notification Service (Amazon SNS).
5. Los suscriptores a ese tema de Amazon SNS reciben el mensaje. En este tutorial, el único suscriptor es una dirección de correo electrónico.

Antes de empezar

En este tutorial se utiliza la AWS Command Line Interface AWS CLI. Si aún no lo ha hecho, siga las instrucciones que figuran en la Guía del usuario de AWS Command Line Interface para instalar y configurar la AWS CLI.

Paso 1: crear una tabla de DynamoDB con un flujo habilitado

En este paso, va a crear una tabla de DynamoDB (`BarkTable`) para almacenar todos los ladridos de los usuarios de Woofer. La clave principal consta de `Username` (clave de partición) y `Timestamp` (clave de ordenación). Ambos atributos son de tipo String.

`BarkTable` tiene una secuencia habilitada. Más adelante en este tutorial, asociará una función de AWS Lambda a la secuencia para crear un disparador.

1. Introduzca el siguiente comando para crear la tabla.

```
aws dynamodb create-table \
    --table-name BarkTable \
    --attribute-definitions AttributeName=Username,AttributeType=S
 AttributeName=Timestamp,AttributeType=S \
    --key-schema AttributeName=Username,KeyType=HASH
 AttributeName=Timestamp,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES
```

2. Busque en los resultados `LatestStreamArn`.

```
...
"LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
stream/timestamp
...
```

   Anote el valor de *`region`* y *`accountID`*, porque los necesitará para los demás pasos de este tutorial.

Paso 2: crear un rol de ejecución para Lambda

En este paso, va a crear un rol de AWS Identity and Access Management (IAM) (`WooferLambdaRole`) y a asignarle permisos. El rol lo utilizará la función de Lambda que va a crear en Paso 4: crear y probar una función Lambda.

Asimismo, va a crear una política para el rol. La política contendrá todos los permisos que la función de Lambda va a necesitar en tiempo de ejecución.

1. Cree un archivo denominado `trust-relationship.json` con el siguiente contenido.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
     {
       "Effect": "Allow",
       "Principal": {
         "Service": "lambda.amazonaws.com"
       },
       "Action": "sts:AssumeRole"
     }
   ]
 }
```

2. Escriba el siguiente comando para crear `WooferLambdaRole`.

```
aws iam create-role --role-name WooferLambdaRole \
    --path "/service-role/" \
    --assume-role-policy-document file://trust-relationship.json
```

3. Cree un archivo denominado `role-policy.json` con el siguiente contenido. (Sustituya *`region`* y *`accountID`* por su región de AWSy su ID de cuenta).

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:us-east-1:111122223333:*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/
BarkTable/stream/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

La política tiene cuatro instrucciones que permiten que `WooferLambdaRole` realice las siguientes acciones:

- Ejecute una función Lambda (`publishNewBark`). Creará la función más adelante en este tutorial.
- Acceder a Amazon CloudWatch Logs. La función de Lambda escribe diagnósticos en CloudWatch Logs en tiempo de ejecución.
- Leer datos de la transmisión de DynamoDB relativos a `BarkTable`.
- Publicar mensajes en Amazon SNS.

4. Introduzca el siguiente comando para asociar la política al `WooferLambdaRole`.

```
aws iam put-role-policy --role-name WooferLambdaRole \
    --policy-name WooferLambdaRolePolicy \
    --policy-document file://role-policy.json
```

Paso 3: crear un tema de Amazon SNS

En este paso, va a crear un tema de Amazon SNS (`wooferTopic`) y suscribe una dirección de correo electrónico a ese tema. La función de Lambda utiliza este tema para publicar los ladridos nuevos de los usuarios de Woofer.

1. Ingrese el siguiente comando para crear un nuevo tema de Amazon SNS.

   ```
   aws sns create-topic --name wooferTopic
   ```

2. Introduzca el siguiente comando para suscribir una dirección de correo electrónico a `wooferTopic`. (Sustituya *`region`* y *`accountID`* por su región de AWS y su ID de cuenta; además, sustituya *`example@example.com`* por una dirección de correo electrónico válida).

   ```
   aws sns subscribe \
       --topic-arn arn:aws:sns:region:accountID:wooferTopic \
       --protocol email \
       --notification-endpoint example@example.com
   ```

3. Amazon SNS envía un mensaje de confirmación a la dirección de correo electrónico indicada. Elija el enlace Confirm subscription (Confirmar suscripción) del mensaje para completar el proceso de suscripción.

Paso 4: crear y probar una función Lambda

En este paso, va a crear una función de AWS Lambda (`publishNewBark`) para procesar los registros de secuencia de `BarkTable`.

La función `publishNewBark` solamente procesa los eventos de la secuencia que corresponden a elementos nuevos de `BarkTable`. La función lee los datos de esos eventos y, a continuación, llama a Amazon SNS para publicarlos.

1. Cree un archivo denominado `publishNewBark.js` con el siguiente contenido. Sustituya *`region`* y *`accountID`* por su región de AWS y su ID de cuenta.

   ```
   'use strict';
   var AWS = require("aws-sdk");
   var sns = new AWS.SNS();

   exports.handler = (event, context, callback) => {
   ```

```
    event.Records.forEach((record) => {
        console.log('Stream record: ', JSON.stringify(record, null, 2));

        if (record.eventName == 'INSERT') {
            var who = JSON.stringify(record.dynamodb.NewImage.Username.S);
            var when = JSON.stringify(record.dynamodb.NewImage.Timestamp.S);
            var what = JSON.stringify(record.dynamodb.NewImage.Message.S);
            var params = {
                Subject: 'A new bark from ' + who,
                Message: 'Woofer user ' + who + ' barked the following at ' + when
 + ':\n\n ' + what,
                TopicArn: 'arn:aws:sns:region:accountID:wooferTopic'
            };
            sns.publish(params, function(err, data) {
                if (err) {
                    console.error("Unable to send message. Error JSON:",
 JSON.stringify(err, null, 2));
                } else {
                    console.log("Results from sending message: ",
 JSON.stringify(data, null, 2));
                }
            });
        }
    });
    callback(null, `Successfully processed ${event.Records.length} records.`);
};
```

2. Cree un archivo zip que contenga `publishNewBark.js`. Si tiene la utilidad de línea de comandos zip, puede introducir el siguiente comando para hacerlo.

```
zip publishNewBark.zip publishNewBark.js
```

3. Al crear la función de Lambda, debe especificar el nombre de recurso de Amazon (ARN) de `WooferLambdaRole`, que creó en Paso 2: crear un rol de ejecución para Lambda. Introduzca el siguiente comando para recuperar este ARN.

```
aws iam get-role --role-name WooferLambdaRole
```

Busque ARN para en los resultados `WooferLambdaRole`.

```
...
"Arn": "arn:aws:iam::region:role/service-role/WooferLambdaRole"
```

```
...
```

Ingrese el siguiente comando para crear la función de Lambda. Sustituya *roleARN* por el ARN de `WooferLambdaRole`.

```
aws lambda create-function \
    --region region \
    --function-name publishNewBark \
    --zip-file fileb://publishNewBark.zip \
    --role roleARN \
    --handler publishNewBark.handler \
    --timeout 5 \
    --runtime nodejs16.x
```

4. Ahora pruebe `publishNewBark` para comprobar que funciona. Para ello, le facilitamos información de entrada que parece un registro auténtico de DynamoDB Streams.

   Cree un archivo denominado `payload.json` con el siguiente contenido. Sustituya *region* y *accountID* por su Región de AWS y su ID de cuenta.

```
{
    "Records": [
        {
            "eventID": "7de3041dd709b024af6f29e4fa13d34c",
            "eventName": "INSERT",
            "eventVersion": "1.1",
            "eventSource": "aws:dynamodb",
            "awsRegion": "region",
            "dynamodb": {
                "ApproximateCreationDateTime": 1479499740,
                "Keys": {
                    "Timestamp": {
                        "S": "2016-11-18:12:09:36"
                    },
                    "Username": {
                        "S": "John Doe"
                    }
                },
                "NewImage": {
                    "Timestamp": {
                        "S": "2016-11-18:12:09:36"
```

```
                    },
                    "Message": {
                        "S": "This is a bark from the Woofer social network"
                    },
                    "Username": {
                        "S": "John Doe"
                    }
                },
                "SequenceNumber": "13021600000000001596893679",
                "SizeBytes": 112,
                "StreamViewType": "NEW_IMAGE"
            },
            "eventSourceARN": "arn:aws:dynamodb:region:account ID:table/BarkTable/
stream/2016-11-16T20:42:48.104"
        }
    ]
}
```

Introduzca el siguiente comando para probar la función de `publishNewBark`.

```
aws lambda invoke --function-name publishNewBark --payload file://payload.json --
cli-binary-format raw-in-base64-out output.txt
```

Si la prueba se realiza correctamente, aparecerá el siguiente resultado.

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

Además, el archivo `output.txt` contendrá el siguiente texto.

```
"Successfully processed 1 records."
```

También recibirá un nuevo mensaje de correo electrónico dentro de unos minutos.

> **Note**
>
> AWS Lambda escribe la información de diagnóstico en Amazon CloudWatch Logs. Si se produce cualquier error en la función Lambda, puede utilizar esos diagnósticos para solucionar el problema:
>
> 1. Abra la consola de CloudWatch en https://console.aws.amazon.com/cloudwatch/.
> 2. En el panel de navegación, elija Logs.
> 3. Elija el grupo de log siguiente: `/aws/lambda/publishNewBark`
> 4. Elija la última secuencia de log para ver el resultado (y los errores) de la función.

Paso 5: crear y probar un disparador

En Paso 4: crear y probar una función Lambda, hemos probado la función Lambda para asegurarnos de que se ejecutaba correctamente. En este paso, va a crear un disparador. Para ello, asociará la función de Lambda (`publishNewBark`) con el origen de eventos (la secuencia `BarkTable`).

1. Al crear el disparador, debe especificar el ARN de la secuencia de `BarkTable`. Introduzca el siguiente comando para recuperar este ARN.

   ```
   aws dynamodb describe-table --table-name BarkTable
   ```

   Busque en los resultados `LatestStreamArn`.

   ```
   ...
    "LatestStreamArn": "arn:aws:dynamodb:region:accountID:table/BarkTable/
   stream/timestamp
   ...
   ```

2. Introduzca el siguiente comando para crear el disparador. Sustituya *`streamARN`* por el ARN de la secuencia real.

   ```
   aws lambda create-event-source-mapping \
       --region region \
       --function-name publishNewBark \
       --event-source streamARN  \
       --batch-size 1 \
   ```

```
    --starting-position TRIM_HORIZON
```

3. Prueba el disparador. Introduzca el siguiente comando para agregar un elemento a `BarkTable`.

```
aws dynamodb put-item \
    --table-name BarkTable \
    --item Username={S="Jane
 Doe"},Timestamp={S="2016-11-18:14:32:17"},Message={S="Testing...1...2...3"}
```

   Debería recibir un nuevo mensaje de correo electrónico dentro de unos minutos.

4. Abra la consola de DynamoDB y agregue algunos elementos más a `BarkTable`. Debe especificar valores para los atributos `Username` y `Timestamp`. También debe especificar un valor para `Message`, aunque no es obligatorio. Debe recibir un nuevo mensaje de correo electrónico por cada elemento que agregue a `BarkTable`.

   La función de Lambda procesa solamente los elementos nuevos que se agregan a `BarkTable`. Si actualiza o elimina un elemento de la tabla, la función no hace nada.

> **Note**
>
> AWS Lambda escribe la información de diagnóstico en Amazon CloudWatch Logs. Si se produce cualquier error en la función Lambda, puede utilizar esos diagnósticos para solucionar el problema.
>
> 1. Abra la consola de CloudWatch en https://console.aws.amazon.com/cloudwatch/.
> 2. En el panel de navegación, elija Logs.
> 3. Elija el grupo de log siguiente: `/aws/lambda/publishNewBark`
> 4. Elija la última secuencia de log para ver el resultado (y los errores) de la función.

Tutorial n.º 2: uso de filtros para procesar algunos eventos con DynamoDB y Lambda

En este tutorial, creará un desencadenador AWS Lambda para procesar solo algunos eventos en un flujo de una tabla de DynamoDB.

Temas

- Resumen global: CloudFormation
- Resumen global: CDK

Con el filtrado de eventos de Lambda puede utilizar expresiones de filtrado para controlar qué eventos envía Lambda a su función para su procesamiento. Puede configurar hasta cinco filtros diferentes por flujos de DynamoDB. Si utiliza intervalos de lotes, Lambda aplica los criterios de filtrado a cada nuevo evento para ver si debe incluirse en el lote actual.

Los filtros se aplican mediante estructuras denominadas `FilterCriteria`. Los 3 atributos principales de `FilterCriteria` son `metadata properties`, `data properties` y `filter patterns`.

A continuación, se muestra un ejemplo de estructura de un evento de DynamoDB Streams:

```
{
  "eventID": "c9fbe7d0261a5163fcb6940593e41797",
  "eventName": "INSERT",
  "eventVersion": "1.1",
  "eventSource": "aws:dynamodb",
  "awsRegion": "us-east-2",
  "dynamodb": {
    "ApproximateCreationDateTime": 1664559083.0,
    "Keys": {
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" }
    },
    "NewImage": {
      "quantity": { "N": "50" },
      "company_id": { "S": "1000" },
      "fabric": { "S": "Florida Chocolates" },
      "price": { "N": "15" },
      "stores": { "N": "5" },
      "product_id": { "S": "1000" },
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" },
      "state": { "S": "FL" },
      "type": { "S": "" }
    },
    "SequenceNumber": "700000000000888747038",
    "SizeBytes": 174,
    "StreamViewType": "NEW_AND_OLD_IMAGES"
  },
  "eventSourceARN": "arn:aws:dynamodb:us-east-2:111122223333:table/chocolate-table-
StreamsSampleDDBTable-LUOI6UXQY7J1/stream/2022-09-30T17:05:53.209"
}
```

Las `metadata properties` son los campos del objeto de evento. En el caso de DynamoDB Streams, las `metadata properties` son campos como `dynamodb` o `eventName`.

Las `data properties` son los campos del cuerpo de evento. Para filtrar las `data properties`, asegúrese de incluirlas en `FilterCriteria` en la clave adecuada. Para los orígenes de eventos de DynamoDB, la clave de datos es `NewImage` u `OldImage`.

Por último, las reglas de filtro definirán la expresión filtro que quiere aplicar a una propiedad específica. Estos son algunos ejemplos:

Operador de comparación	Ejemplo	Sintaxis de reglas (parcial)
Nulo	El tipo de producto es nulo	`{ "product_type": { "S": null } }`
Vacío	El nombre de producto está vacío	`{ "product_name": { "S": [ ""] } }`
Igual a	El estado es igual a Florida	`{ "state": { "S": ["FL"] } }`
Y	El estado del producto es igual a Florida y la categoría del producto es Chocolate	`{ "state": { "S": ["FL"] } , "category": { "S": [ "CHOCOLATE"] } }`
O	El estado del producto es Florida o California	`{ "state": { "S": ["FL","CA"] } }`
No (Negación)	El estado del producto no es Florida	`{"state": {"S": [{"anything-but": ["FL"]}]}}`
Exists	El producto Homemade existe	`{"homemade": {"S": [{"exists": true}]}}`
No existe	El producto Homemade no existe	`{"homemade": {"S": [{"exists": false}]}}`

Operador de comparación	Ejemplo	Sintaxis de reglas (parcial)
Comienza por	PK comienza por COMPANY	`{"PK": {"S": [{"prefix": "COMPANY"}]}}`

Puede especificar hasta cinco patrones de filtrado de eventos para una función Lambda. Observe que cada uno de esos cinco eventos se evaluará como un O lógico. Por lo tanto, si configura dos filtros denominados `Filter_One` y `Filter_Two`, la función Lambda ejecutará `Filter_One` O `Filter_Two`.

> **Note**
>
> En la página de filtrado de eventos de Lambda hay algunas opciones para filtrar y comparar valores numéricos; sin embargo, en el caso de los eventos de filtrado de DynamoDB, esto no se aplica porque los números en DynamoDB se almacenan como cadenas. Por ejemplo `"quantity": { "N": "50" }`, sabemos que es un número debido a la propiedad `"N"`.

Resumen global: CloudFormation

Para mostrar la funcionalidad del filtrado de eventos en la práctica, a continuación se muestra una plantilla de CloudFormation de ejemplo. Esta plantilla generará una tabla de DynamoDB Simple con una clave de partición PK y una clave de clasificación SK con Amazon DynamoDB Streams habilitado. Creará una función Lambda y un rol de ejecución Lambda simple que permitirá escribir registros en Amazon CloudWatch y leer los eventos de Amazon DynamoDB Streams. También agregará la asignación del origen de los eventos entre DynamoDB Streams y la función Lambda, para que la función pueda ejecutarse cada vez que haya un evento en Amazon DynamoDB Streams.

```
AWSTemplateFormatVersion: "2010-09-09"

Description: Sample application that presents AWS Lambda event source filtering
with Amazon DynamoDB Streams.

Resources:
  StreamsSampleDDBTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        - AttributeName: "PK"
```

```
          AttributeType: "S"
        - AttributeName: "SK"
          AttributeType: "S"
      KeySchema:
        - AttributeName: "PK"
          KeyType: "HASH"
        - AttributeName: "SK"
          KeyType: "RANGE"
      StreamSpecification:
        StreamViewType: "NEW_AND_OLD_IMAGES"
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5

  LambdaExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17",
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - lambda.amazonaws.com
            Action:
              - sts:AssumeRole
      Path: "/"
      Policies:
        - PolicyName: root
          PolicyDocument:
            Version: "2012-10-17",
            Statement:
              - Effect: Allow
                Action:
                  - logs:CreateLogGroup
                  - logs:CreateLogStream
                  - logs:PutLogEvents
                Resource: arn:aws:logs:*:*:*
              - Effect: Allow
                Action:
                  - dynamodb:DescribeStream
                  - dynamodb:GetRecords
                  - dynamodb:GetShardIterator
                  - dynamodb:ListStreams
```

```
                Resource: !GetAtt StreamsSampleDDBTable.StreamArn

  EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST

  ProcessEventLambda:
    Type: AWS::Lambda::Function
    Properties:
      Runtime: python3.7
      Timeout: 300
      Handler: index.handler
      Role: !GetAtt LambdaExecutionRole.Arn
      Code:
        ZipFile: |
          import logging

          LOGGER = logging.getLogger()
          LOGGER.setLevel(logging.INFO)

          def handler(event, context):
            LOGGER.info('Received Event: %s', event)
            for rec in event['Records']:
              LOGGER.info('Record: %s', rec)

Outputs:
  StreamsSampleDDBTable:
    Description: DynamoDB Table ARN created for this example
    Value: !GetAtt StreamsSampleDDBTable.Arn
  StreamARN:
    Description: DynamoDB Table ARN created for this example
    Value: !GetAtt StreamsSampleDDBTable.StreamArn
```

Después de implementar esta plantilla de CloudFormation, puede insertar el siguiente elemento de Amazon DynamoDB:

```
{
 "PK": "COMPANY#1000",
```

```
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

Gracias a la sencilla función Lambda incluida en línea en esta plantilla de CloudFormation, verá los eventos en los grupos de registro de Amazon CloudWatch para la función Lambda de la siguiente manera:

```
{
  "eventID": "c9fbe7d0261a5163fcb6940593e41797",
  "eventName": "INSERT",
  "eventVersion": "1.1",
  "eventSource": "aws:dynamodb",
  "awsRegion": "us-east-2",
  "dynamodb": {
    "ApproximateCreationDateTime": 1664559083.0,
    "Keys": {
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" }
    },
    "NewImage": {
      "quantity": { "N": "50" },
      "company_id": { "S": "1000" },
      "fabric": { "S": "Florida Chocolates" },
      "price": { "N": "15" },
      "stores": { "N": "5" },
      "product_id": { "S": "1000" },
      "SK": { "S": "PRODUCT#CHOCOLATE#DARK#1000" },
      "PK": { "S": "COMPANY#1000" },
      "state": { "S": "FL" },
      "type": { "S": "" }
    },
    "SequenceNumber": "700000000000888747038",
    "SizeBytes": 174,
    "StreamViewType": "NEW_AND_OLD_IMAGES"
  },
```

```
  "eventSourceARN": "arn:aws:dynamodb:us-east-2:111122223333:table/chocolate-table-
StreamsSampleDDBTable-LUOI6UXQY7J1/stream/2022-09-30T17:05:53.209"
}
```

Ejemplos de filtro

- Solo productos que coincidan con un estado determinado

Este ejemplo modifica la plantilla de CloudFormation para incluir un filtro que coincida con todos los productos que provienen de Florida, con la abreviatura “FL”.

```
EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      FilterCriteria:
        Filters:
          - Pattern: '{ "dynamodb": { "NewImage": { "state": { "S": ["FL"] } } } }'
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST
```

Una vez que vuelva a implementar la pila, puede agregar el siguiente elemento de DynamoDB a la tabla. Tenga en cuenta que no aparecerá en los registros de la función Lambda, porque el producto de este ejemplo es de California.

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK#1000",
 "company_id": "1000",
 "fabric": "Florida Chocolates",
 "price": 15,
 "product_id": "1000",
 "quantity": 50,
 "state": "CA",
 "stores": 5,
 "type": ""
}
```

- Solo los elementos que comienzan por algunos valores en PK y SK

En este ejemplo se modifica la plantilla de CloudFormation para incluir la siguiente condición:

```
EventSourceDDBTableStream:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: True
      FilterCriteria:
        Filters:
          - Pattern: '{"dynamodb": {"Keys": {"PK": { "S": [{ "prefix":
 "COMPANY" }] },"SK": { "S": [{ "prefix": "PRODUCT" }] }}}}'
      EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
      FunctionName: !GetAtt ProcessEventLambda.Arn
      StartingPosition: LATEST
```

Observe que la condición AND requiere que la condición esté en el patrón, donde las claves PK y SK están en la misma expresión separadas por una coma.

O bien empieza con algunos valores en PK y SK o es de cierto estado.

En este ejemplo se modifica la plantilla de CloudFormation para incluir las siguientes condiciones:

```
 EventSourceDDBTableStream:
   Type: AWS::Lambda::EventSourceMapping
   Properties:
     BatchSize: 1
     Enabled: True
     FilterCriteria:
       Filters:
         - Pattern: '{"dynamodb": {"Keys": {"PK": { "S": [{ "prefix":
"COMPANY" }] },"SK": { "S": [{ "prefix": "PRODUCT" }] }}}}'
         - Pattern: '{ "dynamodb": { "NewImage": { "state": { "S": ["FL"] } } } }'
     EventSourceArn: !GetAtt StreamsSampleDDBTable.StreamArn
     FunctionName: !GetAtt ProcessEventLambda.Arn
     StartingPosition: LATEST
```

Observe que la condición OR se agrega con la incorporación de nuevos patrones en la sección de filtro.

Resumen global: CDK

La siguiente plantilla de formación de proyectos CDK de ejemplo muestra la funcionalidad de filtrado de eventos. Antes de trabajar con este proyecto CDK deberá instalar los requisitos previos, incluida la ejecución de los scripts de preparación.

Crear un proyecto CDK

Primero cree un nuevo proyecto AWS CDK, mediante la invocación de `cdk init` en un directorio vacío.

```
mkdir ddb_filters
cd ddb_filters
cdk init app --language python
```

El comando `cdk init` utiliza el nombre de la carpeta del proyecto para asignar un nombre a varios elementos del proyecto, incluidas las clases, las subcarpetas y los archivos. Los guiones del nombre de la carpeta se convierten en guiones bajos. Por lo demás, el nombre debe seguir el formato de un identificador Python. Por ejemplo, no debe comenzar por un número ni contener espacios.

Para trabajar con el nuevo proyecto, active su entorno virtual. Esto permite que las dependencias del proyecto se instalen localmente en la carpeta del proyecto, en lugar de hacerlo globalmente.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

> **Note**
>
> Es posible que reconozca esto como el comando de Mac/Linux para activar un entorno virtual. Las plantillas de Python incluyen un archivo por lotes, `source.bat`, que permite utilizar el mismo comando en Windows. El comando tradicional de Windows `.venv\Scripts\activate.bat` también funciona. Si ha inicializado su proyecto AWS CDK con AWS CDK Toolkit v1.70.0 o anterior, su entorno virtual se encuentra en el directorio `.env` en lugar de `.venv`.

Infraestructura básica

Abra el archivo `./ddb_filters/ddb_filters_stack.py` en el editor de texto que desee. Este archivo se generó automáticamente al crear el proyecto AWS CDK.

A continuación, agregue las funciones `_create_ddb_table` y `_set_ddb_trigger_function`. Estas funciones crearán una tabla de DynamoDB con la clave de partición PK y la clave de clasificación SK en modo de aprovisionamiento bajo demanda, con Amazon DynamoDB Streams habilitado de forma predeterminada para mostrar las imágenes nueva y antigua.

La función Lambda se almacenará en la carpeta `lambda` debajo del archivo `app.py`. Este archivo se creará más adelante. Incluirá la variable de entorno `APP_TABLE_NAME`, que será el nombre de la tabla de Amazon DynamoDB creada por esta pila. En la misma función concederemos permisos de lectura del flujo a la función Lambda. Por último, se suscribirá a DynamoDB Streams como origen de eventos para la función Lambda.

Al final del archivo en el método `__init__`, llamará a las construcciones respectivas para inicializarlas en la pila. Para proyectos más grandes que requieran componentes y servicios adicionales, podría ser mejor definir estas construcciones fuera de la pila base.

```
import os
import json

import aws_cdk as cdk
from aws_cdk import (
    Stack,
    aws_lambda as _lambda,
    aws_dynamodb as dynamodb,
)
from constructs import Construct


class DdbFiltersStack(Stack):

    def _create_ddb_table(self):
        dynamodb_table = dynamodb.Table(
            self,
            "AppTable",
            partition_key=dynamodb.Attribute(
                name="PK", type=dynamodb.AttributeType.STRING
            ),
            sort_key=dynamodb.Attribute(
                name="SK", type=dynamodb.AttributeType.STRING),
            billing_mode=dynamodb.BillingMode.PAY_PER_REQUEST,
            stream=dynamodb.StreamViewType.NEW_AND_OLD_IMAGES,
            removal_policy=cdk.RemovalPolicy.DESTROY,
        )
```

```
        cdk.CfnOutput(self, "AppTableName", value=dynamodb_table.table_name)
        return dynamodb_table

    def _set_ddb_trigger_function(self, ddb_table):
        events_lambda = _lambda.Function(
            self,
            "LambdaHandler",
            runtime=_lambda.Runtime.PYTHON_3_9,
            code=_lambda.Code.from_asset("lambda"),
            handler="app.handler",
            environment={
                "APP_TABLE_NAME": ddb_table.table_name,
            },
        )

        ddb_table.grant_stream_read(events_lambda)

        event_subscription = _lambda.CfnEventSourceMapping(
            scope=self,
            id="companyInsertsOnlyEventSourceMapping",
            function_name=events_lambda.function_name,
            event_source_arn=ddb_table.table_stream_arn,
            maximum_batching_window_in_seconds=1,
            starting_position="LATEST",
            batch_size=1,
        )

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        ddb_table = self._create_ddb_table()
        self._set_ddb_trigger_function(ddb_table)
```

Ahora crearemos una función Lambda muy sencilla que imprimirá los registros en Amazon CloudWatch. Para ello, cree una carpeta nueva llamada `lambda`.

```
mkdir lambda
touch app.py
```

Con su editor de texto favorito, agregue el siguiente contenido al archivo `app.py`:

```
import logging
```

```
LOGGER = logging.getLogger()
LOGGER.setLevel(logging.INFO)


def handler(event, context):
    LOGGER.info('Received Event: %s', event)
    for rec in event['Records']:
        LOGGER.info('Record: %s', rec)
```

Asegúrese de que está en la carpeta `/ddb_filters/` y escriba el siguiente comando para crear la aplicación de muestra:

```
cdk deploy
```

En algún momento se le pedirá que confirme si desea implementar la solución. Escriba Y para aceptar los cambios.

```
##########################################################################################
# + # ${LambdaHandler/ServiceRole} # arn:${AWS::Partition}:iam::aws:policy/service-
role/AWSLambdaBasicExecutionRole #
##########################################################################################

Do you wish to deploy these changes (y/n)? y

...

#  Deployment time: 67.73s

Outputs:
DdbFiltersStack.AppTableName = DdbFiltersStack-AppTable815C50BC-1M1W7209V5YPP
Stack ARN:
arn:aws:cloudformation:us-east-2:111122223333:stack/
DdbFiltersStack/66873140-40f3-11ed-8e93-0a74f296a8f6
```

Una vez implementados los cambios, abra la consola de AWS y agregue un elemento a la tabla.

```
{
 "PK": "COMPANY#1000",
 "SK": "PRODUCT#CHOCOLATE#DARK",
 "company_id": "1000",
 "type": "",
```

```
 "state": "FL",
 "stores": 5,
 "price": 15,
 "quantity": 50,
 "fabric": "Florida Chocolates"
}
```

Los registros de CloudWatch ahora deben contener toda la información de esta entrada.

Ejemplos de filtro

- Solo productos que coincidan con un estado determinado

Abra el archivo `ddb_filters/ddb_filters/ddb_filters_stack.py` y modifíquelo para incluir el filtro que coincide con todos los productos que son iguales a “FL”. Esto se puede revisar justo debajo de `event_subscription` en la línea 45.

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}
                )
            },
        ]
    },
)
```

- Solo los elementos que comienzan por algunos valores en PK y SK

Modifique el script python para incluir la siguiente condición:

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {
```

```
                        {
                            "dynamodb": {
                                "Keys": {
                                    "PK": {"S": [{"prefix": "COMPANY"}]},
                                    "SK": {"S": [{"prefix": "PRODUCT"}]},
                                }
                            }
                        }
                    }
                )
            },
        ]
    },
```

- O bien comience con algunos valores en PK y SK o desde un determinado estado.

Modifique el script python para incluir las siguientes condiciones:

```
event_subscription.add_property_override(
    property_path="FilterCriteria",
    value={
        "Filters": [
            {
                "Pattern": json.dumps(
                    {
                        {
                            "dynamodb": {
                                "Keys": {
                                    "PK": {"S": [{"prefix": "COMPANY"}]},
                                    "SK": {"S": [{"prefix": "PRODUCT"}]},
                                }
                            }
                        }
                    }
                )
            },
            {
                "Pattern": json.dumps(
                    {"dynamodb": {"NewImage": {"state": {"S": ["FL"]}}}}
                )
            },
        ]
```

```
    },
)
```

Observe que la condición OR se agrega al agregar más elementos a la matriz Filters.

Limpieza

Localice la pila de filtros en la base de su directorio de trabajo y ejecute `cdk destroy`. Se le pedirá que confirme la eliminación del recurso:

```
cdk destroy
Are you sure you want to delete: DdbFiltersStack (y/n)? y
```

Prácticas recomendadas para usar DynamoDB Streams con Lambda

Una función AWS Lambda se ejecuta dentro de un contenedor, a saber, un entorno de ejecución aislado de las demás funciones. Cuando se ejecuta una función por primera vez, AWS Lambda crea un nuevo contenedor y comienza a ejecutar el código de la función.

Una función Lambda posee un controlador que se ejecuta una vez en cada invocación. El controlador contiene la lógica empresarial principal de la función. Por ejemplo, la función Lambda que se muestra en Paso 4: crear y probar una función Lambda tiene un controlador que puede procesar los registros de una transmisión de DynamoDB.

También puede proporcionar código de inicialización que se ejecuta una sola vez después de crear el contenedor, pero antes de que AWS Lambda ejecute el controlador por primera vez. La función Lambda que se muestra en Paso 4: crear y probar una función Lambda incluye código de inicialización que importa el SDK para JavaScript en Node.js y crea un cliente para Amazon SNS. Estos objetos únicamente deben definirse una vez, fuera del controlador.

Después de ejecutar la función, AWS Lambda puede optar por reutilizar el contenedor en invocaciones posteriores de la función. En este caso, el controlador de la función podría volver a utilizar los recursos que se han definido en el código de inicialización. (No se puede controlar durante cuánto tiempo AWS Lambda conservará el contenedor ni si este se reutilizará o no).

Para los desencadenadores de DynamoDB que utilizan AWS Lambda, recomendamos lo siguiente:

- AWSLas instancias de los clientes de servicios de deben crearse en el código de inicialización, no en el controlador. Esto permite que AWS Lambda reutilice las conexiones existentes mientras dure la vida útil del contenedor.

- En general, no es necesario administrar de forma explícita las conexiones ni implementar la agrupación de conexiones, porque AWS Lambda lo hace automáticamente.

Un consumidor de Lambda para un flujo de DynamoDB no garantiza exactamente una entrega y puede generar duplicados. Asegúrese de que el código de la función de Lambda sea idempotente para evitar que surjan problemas inesperados por la aparición de duplicados.

Para obtener más información, consulte Prácticas recomendadas para trabajar con las funciones AWS Lambda en la Guía para desarrolladores de AWS Lambda.

DynamoDB Streams y Apache Flink

Puede consumir registros de Amazon DynamoDB Streams con Apache Flink. Con Amazon Managed Service para Apache Flink, puede transformar y analizar datos de streaming en tiempo real mediante Apache Flink. Apache Flink es un marco de procesamiento de flujos de código abierto para procesar datos en tiempo real. El conector de Amazon DynamoDB Streams para Apache Flink simplifica la creación y la administración de las cargas de trabajo de Apache Flink y permite integrar aplicaciones con otros Servicios de AWS.

Amazon Managed Service para Apache Flink lo ayuda a crear rápidamente aplicaciones de procesamiento de flujos de extremo a extremo para análisis de registros, análisis de flujos de clics, Internet de las cosas (IoT), tecnología publicitaria, juegos y mucho más. Los cuatro casos de uso más comunes son la extracción-transformación-carga (ETL) de streaming, las aplicaciones controladas por eventos, los análisis en tiempo real con capacidad de respuesta y la consulta interactiva de flujos de datos. Para obtener más información sobre la escritura en Apache Flink desde Amazon DynamoDB Streams, consulte Conector de Amazon DynamoDB Streams.

Uso del parámetro ShardFilter con el conector de los flujos de DynamoDB para Apache Flink

Amazon DynamoDB Streams admite el parámetro `ShardFilter` en la API de `DescribeStream` para buscar un subconjunto de particiones. Al especificar una partición principal en la solicitud, los flujos de DynamoDB devolverán sus particiones secundarias inmediatas. Puede usar el parámetro `ShardFilter` para detectar rápidamente las particiones secundarias después de cerrar la partición principal, lo que hace que las aplicaciones de procesamiento de flujos sean más receptivas y rentables.

Amazon Managed Service para Apache Flink admite el parámetro `ShardFilter` al leer desde los flujos de DynamoDB. Para utilizar esta característica, debe solicitar acceso a una versión especial del conector de Flink que incluya soporte de `ShardFilter`. Para empezar, abra un caso de soporte

en el AWS Support Center y solicite acceso a estos binarios. Para obtener más información sobre cómo usar los flujos de DynamoDB con Apache Flink, consulte Lectura y procesamiento de un flujo en la Guía para desarrolladores de DynamoDB.

Aceleración en memoria con DynamoDB Accelerator (DAX)

Amazon DynamoDB está diseñado para facilitar el escalado y mejorar el rendimiento. En la mayoría de los casos, los tiempos de respuesta de DynamoDB pueden medirse en milisegundos con cifras de un solo dígito. No obstante, existen determinados casos de uso que requieren tiempos de respuesta en microsegundos. Para estos casos de uso, DynamoDB Accelerator (DAX) proporciona tiempos de respuesta más breves al acceder a datos de consistencia final.

DAX es un servicio de almacenamiento en caché compatible con DynamoDB que le permite beneficiarse de un rápido rendimiento en memoria para las aplicaciones más exigentes. DAX aborda tres escenarios principales:

1. Como caché en memoria, DAX reduce los tiempos de respuesta de las cargas de trabajo de lectura eventualmente consistente en un orden de magnitud, de cifras en milisegundos de un solo dígito a microsegundos.
2. DAX reduce la complejidad operativa y de las aplicaciones al ofrecer un servicios administrado que es compatible mediante API con DynamoDB. Por consiguiente, solo requiere unos mínimos cambios funcionales para poder usarlo con una aplicación existente.
3. Para las cargas de trabajo intensas o en ráfagas, DAX ofrece mayor rendimiento y posibles ahorros en el coste, porque disminuye la necesidad de aprovisionar unidades de capacidad de lectura de más. Esto resulta especialmente beneficioso para las aplicaciones que requieren lecturas reiteradas de claves individuales.

DAX es compatible con el cifrado en el lado del servidor. Con el cifrado en reposo, los datos conservados por DAX en el disco serán cifrados. DAX escribe datos en disco como parte de la propagación de cambios desde el nodo principal a las réplicas de lectura. Para obtener más información, consulte Cifrado en reposo de DAX.

DAX también admite el cifrado en tránsito, lo que garantiza que todas las solicitudes y respuestas entre la aplicación y el clúster estén cifradas por seguridad de nivel de transporte (TLS), y las conexiones al clúster se pueden autenticar mediante la verificación de un certificado x509 del clúster. Para obtener más información, consulte Cifrado en tránsito de DAX.

Temas

- Casos de uso de DAX
- Notas de uso de DAX

- DAX: cómo funciona
- Componentes del clúster de DAX
- Creación de un clúster de DAX
- Modelos de coherencia de DAX y DynamoDB
- Desarrollo con el cliente de DynamoDB Accelerator (DAX)
- Administración de los clústeres de DAX
- Supervisión de DynamoDB Accelerator
- Instancias ampliables DAX T3/T2
- Control de acceso a DAX
- Cifrado en reposo de DAX
- Cifrado en tránsito de DAX
- Uso de roles de IAM vinculados a servicios para DAX
- Acceso a DAX a través de las cuentas de AWS
- Guía de tamaño del clúster de DAX

Casos de uso de DAX

DAX proporciona acceso a datos de consistencia final de tablas de DynamoDB con una latencia de microsegundos. Un clúster Multi-AZ de DAX puede atender millones de solicitudes por segundo.

DAX es ideal para los siguientes tipos de aplicaciones:

- Las aplicaciones que requieren realizar lecturas con la máxima rapidez de respuesta posible. Algunos ejemplos de ello son las aplicaciones de subastas en tiempo real, juegos sociales o de transacciones comerciales. DAX proporciona un rápido rendimiento de lectura en memoria para estos casos de uso.
- Las aplicaciones que leen una pequeña cantidad de elementos con más frecuencia que otros. Por ejemplo, tomemos un sistema de comercio electrónico (e-commerce) que durante un día promociona un producto popular a menos precio. Durante la promoción, la demanda de ese producto (y de sus datos en DynamoDB) experimentaría un aumento drástico en comparación con todos los demás productos. Para mitigar los efectos de una clave "caliente" y de una distribución desigual del tráfico, podría delegar la actividad de lectura en una caché de DAX hasta que finalice el día de promoción.

- Las aplicaciones con gran intensidad de lectura sujetas a requisitos de costos. Con DynamoDB, usted aprovisiona la cantidad de lecturas por segundo que la aplicación requiere. Si la actividad de lectura aumenta, puede incrementar el desempeño de lectura provisionado de la tabla (lo que supone un costo adicional). Si lo prefiere, puede delegar la actividad de la aplicación a un clúster de DAX y reducir la cantidad de unidades de capacidad de lectura que, de no hacerlo, tendría que adquirir.
- Las aplicaciones que requieren realizar lecturas reiteradas de un conjunto de datos voluminoso. Una aplicación de este tipo podría desviar recursos de base de datos de otras aplicaciones. Por ejemplo, un análisis prolongado de los datos meteorológicos regionales podría consumir temporalmente toda la capacidad de lectura de una tabla de DynamoDB. Esto tendría un efecto negativo en otras aplicaciones que necesitasen obtener acceso a los mismos datos. En cambio, con DAX, el análisis de los datos meteorológicos podría llevarse a cabo usando datos almacenados en caché.

DAX no es ideal para los siguientes tipos de aplicaciones:

- Las aplicaciones que requieran lecturas de consistencia alta (o que no toleran las lecturas consistentes finales).
- Las aplicaciones que no requieran tiempos de respuesta en microsegundos para lecturas o que no necesiten delegar la actividad de lectura reiterada de tablas subyacentes.
- Aplicaciones que requieren mucho tráfico de escritura. Un alto volumen de escrituras aumenta la replicación en los nodos de DAX de un clúster. Esto provoca un aumento del consumo de recursos y el riesgo de problemas de disponibilidad.
- Aplicaciones sin muchas lecturas repetidas. DAX funciona mejor cuando las tasas de aciertos de caché superan el 90 %. Las tasas de aciertos de caché más bajas aumentan los errores de caché, lo que consume más recursos en todo el clúster de DAX.

Notas de uso de DAX

- Para obtener una lista de regiones de AWS en las que DAX está disponible, consulte Precios de Amazon DynamoDB.
- DAX admite aplicaciones escritas en Go, Java, Node.js, Python y .NET utilizando los clientes proporcionados por AWS para dichos lenguajes de programación.
- DAX solo está disponible para la plataforma EC2-VPC.

- La política de la función del servicio de clúster de DAX debe permitir la acción `dynamodb:DescribeTable` para mantener los metadatos sobre la tabla de DynamoDB.
- Los clústeres de DAX conservan metadatos sobre los nombres de atributo de los elementos que almacenan. Dichos metadatos se conservan indefinidamente (incluso después de que el elemento haya vencido o se haya desalojado de la caché). Las aplicaciones que utilizan un número ilimitado de nombres de atributo pueden, con el tiempo, agotar la memoria en el clúster de DAX. Esta limitación se aplica solo a los nombres de atributo de nivel superior, no a los nombres de atributo anidados. Ejemplos de nombres de atributo de nivel superior problemáticos son las marcas temporales, los UUID y los ID de sesión.

  Esta limitación solo se aplica a los nombres de atributo, no a sus valores. Los elementos como el siguiente no son un problema.

  ```
  {
      "Id": 123,
      "Title": "Bicycle 123",
      "CreationDate": "2017-10-24T01:02:03+00:00"
  }
  ```

  Pero los elementos como el siguiente sí lo son, si hay suficientes y tienen cada uno de ellos una marca temporal diferente.

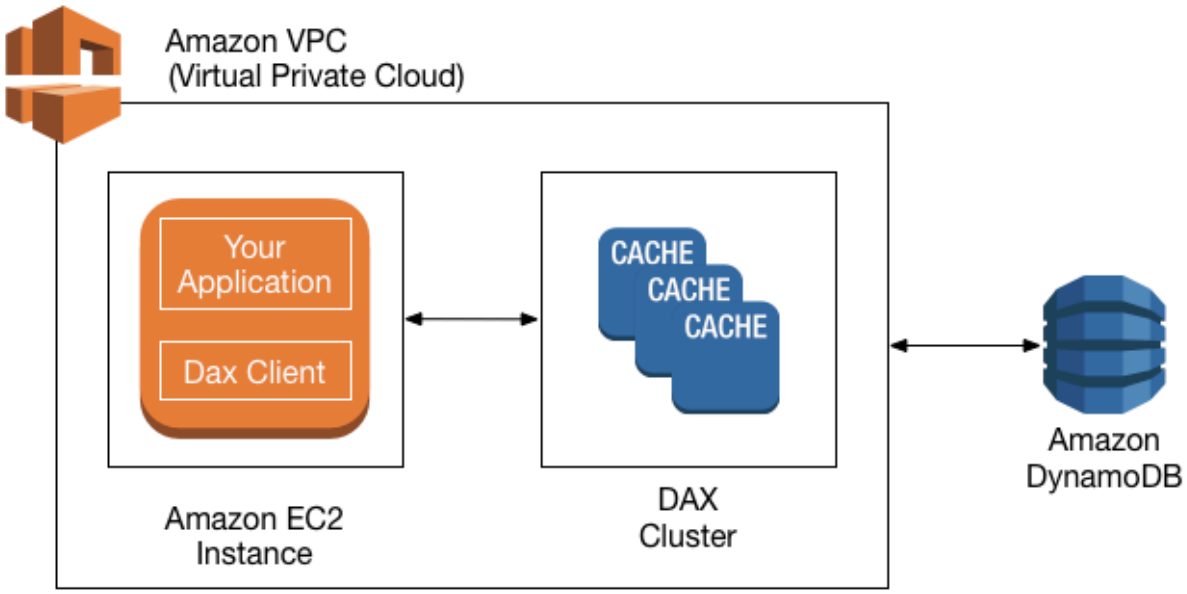
  ```
  {
      "Id": 123,
      "Title": "Bicycle 123",
      "2017-10-24T01:02:03+00:00": "created"
  }
  ```

DAX: cómo funciona

Amazon DynamoDB Accelerator (DAX) está diseñado para ejecutarse en un entorno Amazon Virtual Private Cloud (Amazon VPC). El servicio Amazon VPC define una red virtual que se parece mucho a un centro de datos tradicional. Con una VPC, puede controlar el rango de direcciones IP, las subredes, las tablas de ruteo, las gateways de red y los ajustes de seguridad. Puede lanzar un clúster de DAX en la red virtual y controlar el acceso al clúster mediante grupos de seguridad de Amazon VPC.

> **Note**
>
> Si creó la cuenta de AWS después del 4 de diciembre de 2013, ya dispone de una VPC predeterminada en cada región de AWS. La VPC está lista para comenzar a usarla de inmediato sin tener que realizar ningún paso de configuración adicional.
> Para obtener más información acerca de las VPC predeterminadas, consulte VPC predeterminada y subredes predeterminadas en la Guía del usuario de Amazon VPC.

En el siguiente diagrama se muestra información general de alto nivel sobre DAX.



Para crear un clúster de DAX, se usa la Consola de administración de AWS. A menos que especifique otra cosa, el clúster de DAX se ejecuta dentro de la VPC predeterminada. Para ejecutar su aplicación, lance una instancia de Amazon EC2 en su Amazon VPC. A continuación, implemente su aplicación (con el cliente de DAX) en la instancia EC2.

En tiempo de ejecución, el cliente de DAX dirige todas las solicitudes de la API de DynamoDB de la aplicación al clúster de DAX. Si DAX puede procesar una de estas solicitudes de la API directamente, lo hace. En caso contrario, transmite la solicitud a DynamoDB.

Por último, el clúster de DAX devuelve los resultados a la aplicación.

Temas

- Cómo procesa DAX las solicitudes
- Caché de elementos
- Caché de consultas

Cómo procesa DAX las solicitudes

Un clúster de DAX consta de uno o varios nodos. Cada nodo ejecuta su propia instancia del software de almacenamiento en caché de DAX. Uno de los nodos sirve como nodo principal del clúster. Los nodos adicionales (en caso de incluirse) actúan como réplicas de lectura. Para obtener más información, consulte Nodos.

La aplicación puede acceder a DAX especificando el punto de enlace del clúster de DAX. El software de cliente de DAX utiliza el punto de enlace del clúster para realizar el ruteo y el equilibrio de carga inteligente.

Operaciones de lectura

DAX puede responder a las siguientes llamadas a la API:

- `GetItem`
- `BatchGetItem`
- `Query`
- `Scan`

Si la solicitud especifica lecturas eventualmente consistentes (el comportamiento predeterminado), intenta leer el elemento en DAX:

- Si el elemento está disponible en DAX (un acierto de caché), DAX devuelve el elemento a la aplicación sin acceder a DynamoDB.
- Si el elemento no está disponible en DAX (un error de caché), DAX transmite la solicitud a DynamoDB. Cuando recibe la respuesta de DynamoDB, DAX devuelve los resultados a la aplicación. También escribe los resultados en la caché en el nodo principal.

> **Note**
>
> Si el clúster contiene réplicas de lectura, DAX las mantiene automáticamente sincronizadas con el nodo primario. Para obtener más información, consulte Clústeres.

Si la solicitud especifica lecturas fuertemente consistentes, DAX transmite la solicitud a DynamoDB. Los resultados de DynamoDB no se almacenan en caché en DAX. Sencillamente, se devuelven a la aplicación.

Operaciones de escritura

Las siguientes operaciones de la API de DAX se consideran de "escritura indirecta" (write-through):

- `BatchWriteItem`
- `UpdateItem`
- `DeleteItem`
- `PutItem`

Con estas operaciones, los datos se escriben primero en la tabla de DynamoDB y, a continuación, en el clúster de DAX. La operación solo se lleva a cabo correctamente si los datos se escriben correctamente en ambos casos: en la tabla y en DAX.

Otras operaciones

DAX no reconoce ninguna operación de DynamoDB de administración de tablas (tales como `CreateTable`, `UpdateTable`, etc.). Si la aplicación tiene que realizar estas operaciones, debe obtener acceso a DynamoDB directamente, sin usar DAX.

Para obtener información detallada acerca de la coherencia de DAX y DynamoDB, consulte Modelos de coherencia de DAX y DynamoDB.

Para obtener información sobre el funcionamiento de las transacciones en DAX, consulte Uso de las API transaccionales en DynamoDB Accelerator (DAX).

Limitación de velocidad de solicitudes

Si el número de solicitudes enviadas a DAX supera la capacidad de un nodo, DAX limita la velocidad a la que acepta solicitudes adicionales devolviendo una ThrottlingException. DAX evalúa

continuamente la utilización de la CPU para determinar el volumen de solicitudes que puede procesar manteniendo el clúster en buen estado.

Puede monitorear la métrica ThrottledRequestCount que DAX publica en Amazon CloudWatch. Si ve estas excepciones periódicamente, debería plantearse escalar el clúster.

Caché de elementos

DAX mantiene una caché de elementos para almacenar los resultados de las operaciones `GetItem` y `BatchGetItem`. Los elementos en la caché representan datos con consistencia final de DynamoDB y se almacenan según sus valores de clave principal.

Cuando una aplicación envía una solicitud `GetItem` o `BatchGetItem`, DAX intenta leer los elementos directamente en la caché de elementos, para lo cual utiliza los valores de clave especificados. Si encuentra los elementos (aciertos de caché), DAX los devuelve a la aplicación de forma inmediata. Si los elementos no se encuentran (error de caché), DAX envía la solicitud a DynamoDB. DynamoDB procesa las solicitudes usando lecturas eventualmente consistentes y devuelve los elementos a DAX. DAX los almacena en la caché de elementos y, a continuación, los devuelve a la aplicación.

La caché de elementos tiene una configuración de período de vida (TTL, por sus siglas en inglés), predeterminada de 5 minutos. DAX asigna una marca temporal a cada elemento que escribe en la caché de elementos. Un elemento vence cuando ha permanecido en la caché más tiempo del indicado en el ajuste de TTL. Si emite una solicitud `GetItem` para un elemento vencido, se considera un error de caché, en cuyo caso DAX envía la solicitud `GetItem` a DynamoDB.

> **Note**
>
> Puede especificar el ajuste de TTL de la caché de elementos al crear un clúster de DAX nuevo. Para obtener más información, consulte Administración de los clústeres de DAX .

DAX también mantiene una lista de elementos menos usados recientemente (LRU) para la caché de elementos. La lista LRU rastrea cuándo se escribió un elemento por primera vez en la caché y cuándo se leyó por última vez en ella. Si la caché de elementos se llena, DAX expulsa los elementos más antiguos (aunque todavía no hayan caducado) para dejar espacio a los nuevos. El algoritmo LRU siempre está habilitado para la caché de elementos y no es configurable por el usuario.

Si especifica la configuración de TTL de la caché de elemento en cero, los elementos de la caché de elementos solo se actualizarán debido a una expulsión de LRU o a una operación de «escritura indirecta».

Para obtener información detallada sobre la coherencia de la caché de elementos en DAX, consulte Comportamiento de la caché de elementos de DAX.

Caché de consultas

DAX también mantiene una caché de consultas para almacenar los resultados de las operaciones `Query` y `Scan`. Los elementos de esta caché representan los conjuntos de resultados de las consultas y los exámenes de las tablas de DynamoDB. Estos conjuntos de resultados se almacenan por los valores de sus parámetros.

Cuando una aplicación envía una solicitud `Query` o `Scan`, DAX intenta leer un conjunto de resultados coincidente en la caché de consultas, para lo cual utiliza los valores de los parámetros especificados. Si encuentra el conjunto de resultados (acierto de caché), DAX lo devuelve a la aplicación de forma inmediata. Si el conjunto de resultados no se encuentra (error de caché), DAX envía la solicitud a DynamoDB. DynamoDB procesa las solicitudes usando lecturas eventualmente consistentes y devuelve el conjunto de resultados a DAX. DAX lo almacena en la caché de consultas y, a continuación, lo devuelve a la aplicación.

> **Note**
>
> Puede especificar el ajuste de TTL de la caché de consultas al crear un clúster de DAX nuevo. Para obtener más información, consulte Administración de los clústeres de DAX .

DAX también mantiene una lista LRU para la caché de consultas. La lista rastrea cuándo se escribió un conjunto de resultados por primera vez en la caché y cuándo se leyó por última vez en ella. Si la caché de consultas se llena, DAX expulsa los conjuntos de resultados más antiguos (aunque todavía no hayan vencido) para dejar espacio a los nuevos. El algoritmo LRU siempre está habilitado para la caché de consultas y no es configurable por el usuario.

Si especifica la configuración de TTL de la caché de consulta en cero, la respuesta de la consulta no se almacenará en la caché.

Para obtener información detallada acerca de la coherencia de la caché de consultas en DAX, consulte Comportamiento de la caché de consulta de DAX.

Componentes del clúster de DAX

Un clúster de Amazon DynamoDB Accelerator (DAX) consta de componentes de la infraestructura de AWS. En esta sección se describen estos componentes y cómo funcionan entre sí.

Temas

- Nodos
- Clústeres
- Regiones y zonas de disponibilidad
- Grupos de parámetros
- Grupos de seguridad
- ARN del clúster
- Punto de conexión de clúster
- Puntos de conexión del nodo
- Grupos de subredes
- Eventos
- Periodo de mantenimiento

Nodos

Un nodo es el bloque de creación más pequeño de un clúster de DAX. Cada nodo ejecuta una instancia del software DAX y mantiene una única réplica de los datos almacenados en caché.

Puede escalar el clúster de DAX de una de las dos formas siguientes:

- Agregando más nodos al clúster. Con ello aumenta el desempeño de lectura general del clúster.
- Utilizando un tipo de nodo mayor. Los tipos nodos mayores proporcionan más capacidad y pueden aumentar el desempeño. (Debe crear un nuevo clúster con el nuevo tipo de nodo).

Todos los nodos contenidos en un clúster son del mismo tipo y ejecutan el mismo software de almacenamiento en caché de DAX. Para obtener una lista de tipos de nodos disponibles, consulte Precios de Amazon DynamoDB.

Clústeres

Un clúster es una agrupación lógica de uno o varios nodos que DAX administra como una unidad. Uno de los nodos del clúster se designa como nodo principal y los demás (si los hay), son las réplicas de lectura.

El nodo principal es responsable de lo siguiente:

- Responder a las solicitudes de datos en caché que formulan las aplicaciones.
- Transmitir las operaciones de escritura a DynamoDB.
- Expulsar los datos de la caché, de acuerdo con la política de expulsión del clúster.

Cuando se producen cambios en los datos almacenados en caché en el nodo principal, DAX propaga los cambios a todos los nodos de réplica de lectura mediante registros de replicación. Una vez recibida la confirmación de todas las réplicas de lectura, DynamoDB elimina los registros de replicación del nodo principal.

Un clúster de DAX admite hasta 11 nodos por clúster (el nodo principal y un máximo de 10 réplicas de lectura).

Las réplicas de lectura son responsables de lo siguiente:

- Responder a las solicitudes de datos en caché que formulan las aplicaciones.
- Expulsar los datos de la caché, de acuerdo con la política de expulsión del clúster.

Sin embargo, al contrario que el nodo primario, las réplicas de lectura no escriben en DynamoDB.

Las réplicas de lectura cumplen dos funciones adicionales:

- Escalabilidad. Si tiene gran cantidad de clientes de aplicación que deben obtener acceso a DAX simultáneamente, puede agregar más réplicas para escalar la lectura. DAX distribuirá la carga de forma uniforme entre todos los nodos del clúster. (Otra forma de aumentar el desempeño es utilizar tipos de nodos de caché mayores).
- Alta disponibilidad En el caso de que se produzca un error en el nodo primario, DAX automáticamente realiza una conmutación por error a una réplica de lectura y la designa como nuevo nodo primario. Si se produce un error en un nodo de réplica, otros nodos del clúster de DAX pueden seguir atendiendo las solicitudes hasta que se consigue recuperar el nodo defectuoso. Para disfrutar de la máxima tolerancia a errores, debe implementar réplicas de lectura en zonas de

disponibilidad distintas. Esta configuración garantiza que el clúster de DAX continúe funcionando incluso si una zona de disponibilidad completa deja de estar disponible.

> **Important**
>
> Para su uso en producción, se recomienda encarecidamente utilizar DAX con al menos tres nodos, en donde cada nodo se coloca en zonas de disponibilidad diferentes. Se requieren tres nodos para que un clúster de DAX sea tolerante a errores.
> Un clúster de DAX se puede implementar con uno o dos nodos para cargas de trabajo de desarrollo o pruebas. Los clústeres de uno o dos nodos no son tolerantes a errores y no se recomiendan menos de tres nodos para usarse en producción. Si los clústeres de uno o dos nodos tienen errores de software o hardware, el clúster podría dejar de estar disponible o podrían perderse los datos almacenados en caché.

> **Important**
>
> Un clúster de DAX admite un máximo de 500 tablas de DynamoDB. Si supera las 500 tablas, es posible que se produzca un deterioro de la disponibilidad y el rendimiento del clúster.

Regiones y zonas de disponibilidad

Un clúster de DAX de una región de AWS solo puede interaccionar con las tablas de DynamoDB que se encuentran en esa misma región. Por este motivo, debe asegurarse de lanzar el clúster de DAX en la región correcta. Si tiene tablas de DynamoDB en otras regiones, debe lanzar clústeres de DAX en esas regiones también.

Cada región de se ha diseñado para que se encuentre totalmente aislada de las demás regiones de . Dentro de cada región hay varias zonas de disponibilidad. Al lanzar los nodos en zonas de disponibilidad diferentes, puede lograr la máxima tolerancia a errores.

> **Important**
>
> No coloque todos los nodos del clúster en la misma zona de disponibilidad. En una configuración así, el clúster de DAX deja de estar disponible si se produce un error en la zona de disponibilidad.

Para su uso en producción, se recomienda encarecidamente utilizar DAX con al menos tres nodos, en donde cada nodo se coloca en zonas de disponibilidad diferentes. Se requieren tres nodos para que un clúster de DAX sea tolerante a errores.
Un clúster de DAX se puede implementar con uno o dos nodos para cargas de trabajo de desarrollo o pruebas. Los clústeres de uno o dos nodos no son tolerantes a errores y no se recomiendan menos de tres nodos para su uso en producción. Si los clústeres de uno o dos nodos tienen errores de software o hardware, el clúster podría dejar de estar disponible o podrían perderse los datos almacenados en caché.

Grupos de parámetros

Los grupos de parámetros se utilizan para administrar los ajustes de tiempo de ejecución de los clústeres de DAX. DAX tiene varios parámetros que puede utilizar para optimizar el rendimiento (por ejemplo, puede definir una política de TTL para los datos almacenados en caché). Un grupo de parámetros es un conjunto de parámetros con nombre que se pueden aplicar a un clúster. De este modo, se garantiza que todos los nodos de ese clúster estén configurados exactamente de la misma forma.

Grupos de seguridad

Un clúster de DAX se ejecuta en un entorno de Amazon Virtual Private Cloud (Amazon VPC). Este entorno es una red virtual que está dedicada a su cuenta de AWS y está aislada de las demás VPC. Un grupo de seguridad funciona como un firewall virtual para la VPC que le permite para controlar el tráfico de red entrante y saliente.

Al lanzar un clúster en la VPC, puede agregar una regla de entrada al grupo de seguridad para que permita el tráfico procedente de la red. La regla de entrada especifica el protocolo (TCP) y el número de puerto (8111) del clúster. Después de agregar esta regla al grupo de seguridad, las aplicaciones que se ejecutan dentro de la VPC podrán acceder al clúster de DAX.

ARN del clúster

A cada clúster de DAX se le asigna un nombre de recurso de Amazon (ARN). El formato del ARN es el siguiente.

```
arn:aws:dax:region:accountID:cache/clusterName
```

Utilice el ARN del clúster en la política de IAM para definir los permisos para las operaciones de la API de DAX. Para obtener más información, consulte Control de acceso a DAX.

Punto de conexión de clúster

Cada clúster de DAX proporciona un punto de enlace de clúster para que la aplicación lo utilice. Al obtener acceso al clúster a través de este punto de enlace, la aplicación no tiene que conocer los nombres de los hosts ni los números de los puertos de los nodos individuales del clúster. Su aplicación "conoce" automáticamente todos los nodos del clúster, aunque se agreguen o eliminen réplicas de lectura.

A continuación, se muestra un ejemplo de punto de enlace de un clúster en la región us-east-1 que no está configurado para utilizar el cifrado en tránsito.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

A continuación, se muestra un ejemplo de punto de enlace de un clúster en la misma región que está configurado para utilizar el cifrado en tránsito.

```
daxs://my-encrypted-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Puntos de conexión del nodo

Cada uno de los nodos individuales de un clúster de DAX tiene su propio nombre de host y número de puerto. A continuación, se muestra un ejemplo de punto de enlace de un nodo.

```
myDAXcluster-a.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111
```

La aplicación puede obtener acceso directamente a un nodo utilizando su punto de enlace. Sin embargo, recomendamos tratar el clúster de DAX como una sola unidad y acceder a él mediante el punto de enlace de clúster. El punto de enlace de clúster aísla la aplicación para que no tenga que mantener una lista de nodos ni actualizarla cada vez que se agreguen o eliminen nodos del clúster.

Grupos de subredes

El acceso a los nodos del clúster de DAX está restringido a las aplicaciones que se ejecutan en las instancias de Amazon EC2 en un entorno de Amazon VPC. Puede utilizar grupos de subredes para conceder acceso al clúster desde las instancias de Amazon EC2 que se ejecutan en subredes concretas. Un grupo de subredes es una colección de subredes (que suelen ser privadas) que puede designar para los clústeres que se ejecutan en un entorno de Amazon VPC.

Al crear un clúster de DAX, debe especificar un grupo de subredes. DAX utiliza dicho grupo de subredes para seleccionar una subred y direcciones IP pertenecientes a ella, y asociárselas a los nodos.

Eventos

DAX registra los eventos significativos que tienen lugar en los clústeres, tales como los errores al agregar un nodo, las adiciones de nodos correctas o los cambios en los grupos de seguridad. Puede monitorizar los eventos clave para conocer el estado actual de los clústeres y, según el evento, adoptar medidas correctivas. Puede acceder a estos eventos desde la Consola de administración de AWS o utilizando la acción `DescribeEvents` en la API de administración de DAX.

También puede solicitar que se envíen notificaciones a un tema específico de Amazon Simple Notification Service (Amazon SNS). De ese modo, sabrá de inmediato cuando se produce un evento en su clúster de DAX.

Periodo de mantenimiento

Cada clúster tiene un periodo de mantenimiento semanal para aplicar los cambios del sistema A medida que los cambios se aplican secuencialmente, se reemplaza un nodo existente y se agrega al clúster un nodo nuevo con los cambios aplicados. Durante este periodo, es posible que la aplicación experimente errores transitorios o limitaciones. Por lo tanto, le recomendamos que programe el periodo de mantenimiento durante el tiempo de uso más bajo y que ajuste este programa regularmente como sea necesario. Puede especificar un intervalo de tiempo de hasta 24 horas durante las cuales deban llevarse a cabo todas las actividades de mantenimiento que solicite.

Si no especifica un periodo de mantenimiento preferido al crear o modificar un clúster de caché, DAX asigna un periodo de mantenimiento de 60 minutos un día de la semana al azar. Este periodo de mantenimiento de 60 minutos se selecciona al azar dentro de un bloque de 8 horas por cada Región de AWS. En la siguiente tabla, se muestran los bloques de tiempo de cada región desde los que se asignan los periodos predeterminados de mantenimiento.

Código de región	Nombre de la región	Periodo de mantenimiento
ap-northeast-1	Región Asia-Pacífico (Tokio)	13:00-21:00 UTC
ap-southeast-1	Región Asia-Pacífico (Singapur)	14:00-22:00 UTC

Código de región	Nombre de la región	Periodo de mantenimiento
ap-southeast-2	Región Asia-Pacífico (Sídney)	12:00-20:00 UTC
ap-south-1	Región Asia-Pacífico (Mumbai)	17:30 — 01:30 UTC
cn-northwest-1	Región China (Ningxia)	23:00-07:00 UTC
cn-north-1	Región China (Pekín)	14:00-22:00 UTC
eu-central-1	Región de Europa (Fráncfort)	23:00-07:00 UTC
eu-north-1	Región Europa (Estocolmo)	01:00-09:00 UTC
eu-south-2	Región Europa (España)	21:00 — 05:00 UTC
eu-west-1	Región de Europa (Irlanda)	22:00-06:00 UTC
eu-west-2	Región de Europa (Londres)	23:00-07:00 UTC
eu-west-3	Región Europa (París)	23:00-07:00 UTC
sa-east-1	Región de América del Sur (São Paulo)	01:00 — 09:00 UTC
us-east-1	Región del este de EE. UU. (Norte de Virginia)	03:00-11:00 UTC
us-east-2	Región del este de EE. UU. (Ohio)	23:00-07:00 UTC
us-west-1	Región Oeste de EE. UU. (Norte de California)	06:00 — 17:00 UTC
us-west-2	Región del Oeste de EE. UU (Oregón)	06:00-14:00 UTC

Creación de un clúster de DAX

Esta sección lo guía paso a paso durante el proceso de configuración y el uso de Amazon DynamoDB Accelerator (DAX) por primera vez en el entorno de Amazon Virtual Private Cloud (Amazon VPC) predeterminado. Puede crear su primer clúster de DAX mediante la AWS Command Line Interface (AWS CLI) o la Consola de administración de AWS.

Una vez que haya creado el clúster de DAX, podrá acceder a él desde una instancia de Amazon EC2 que se ejecute en la misma VPC. A partir de ese momento, podrá utilizar el clúster de DAX con un programa de aplicación. Para obtener más información, consulte Desarrollo con el cliente de DynamoDB Accelerator (DAX).

Temas

- Creación de un rol de servicio de IAM para que DAX obtenga acceso a DynamoDB
- DAX e IPv6
- Creación de un clúster de DAX mediante la AWS CLI
- Creación de un clúster de DAX mediante la Consola de administración de AWS

Creación de un rol de servicio de IAM para que DAX obtenga acceso a DynamoDB

Para que el clúster de DAX acceda a las tablas de DynamoDB en su nombre, tendrá que crear una función del servicio. Una función del servicio es un rol de AWS Identity and Access Management (IAM) que autoriza a un servicio de AWS para actuar en su nombre. La función del servicio permitirá a DAX a acceder a las tablas de DynamoDB como si usted estuviese abriéndolas directamente. Debe crear la función del servicio para poder crear el clúster de DAX.

Si utiliza la consola, el flujo de trabajo para crear un clúster verifica si ya existe una función del servicio de DAX. Si no encuentra ninguno, la consola crea un nuevo rol de servicio en su nombre. Para obtener más información, consulte the section called “Paso 2: crear un clúster de DAX”.

Si utiliza la AWS CLI, tendrá que especificar una función del servicio de DAX creado previamente. De lo contrario, deberá crear un nuevo rol de servicio de antemano. Para obtener más información, consulte Paso 1: crear un rol de servicio de IAM para que DAX obtenga acceso a DynamoDB mediante la AWS CLI.

Permisos necesarios para crear un rol de servicio

La política `AdministratorAccess` administrada por AWS proporciona todos los permisos que se necesitan para crear un clúster de DAX y una función del servicio. Si su usuario tiene asociado `AdministratorAccess`, no tiene que hacer nada más.

De lo contrario, deberá agregar los siguientes permisos a la política de IAM para que el usuario pueda crear el rol de servicio:

- `iam:CreateRole`
- `iam:CreatePolicy`
- `iam:AttachRolePolicy`
- `iam:PassRole`

Asocie estos permisos al usuario que intenta realizar la acción.

> **Note**
>
> Los permisos `iam:CreateRole`, `iam:CreatePolicy`, `iam:AttachRolePolicy` y `iam:PassRole` no se incluyen en las políticas administradas por AWS para DynamoDB. Esto es así por diseño, ya que estos permisos permitirían escalar los privilegios. Es decir, un usuario podría utilizarlos para crear una nueva política de administrador y asociarla a un rol existente. Por este motivo, usted (el administrador de su clúster de DAX) debe agregar explícitamente estos permisos a la política.

Solución de problemas

Si la política de usuario no incluye los permisos `iam:CreateRole`, `iam:CreatePolicy` e `iam:AttachPolicy`, se producirán mensajes de error. En la tabla siguiente se muestran estos mensajes y se describe cómo corregir los problemas.

Si aparece este mensaje de error...	Haga lo siguiente:
`User: arn:aws:iam:: accountID :user/userName is not authorized to perform: iam:CreateRole on`	Agregue `iam:CreateRole` a la política de usuario.

Si aparece este mensaje de error...	Haga lo siguiente:
`resource: arn:aws:iam:: accountID :role/service-role/ roleName`	
`User: arn:aws:iam:: accountID :user/userName is not authorize d to perform: iam:CreatePolicy on resource: policy policyName`	Agregue `iam:CreatePolicy` a la política de usuario.
`User: arn:aws:iam:: accountID :user/userName is not authorized to perform: iam:AttachRolePolicy on resource: role daxServiceRole`	Agregue `iam:AttachRolePolicy` a la política de usuario.

Para obtener más información sobre las políticas de IAM que se requieren para administrar clústeres de DAX, consulte Control de acceso a DAX.

DAX e IPv6

DynamoDB DAX ahora admite direccionamiento IPv6, que le permite crear clústeres que operan en modos de red de solo IPv4, solo IPv6 o doble pila. Esto ayuda a mejorar las capacidades de red para cumplir con los requisitos de infraestructura en constante evolución.

Tipos de red:

Puede crear clústeres con los siguientes tipos de red:

- Solo IPv4
- Solo IPv6
- Doble pila (admite IPv4 e IPv6)

Características clave de:

Con compatibilidad con IPv6, puede hacer lo siguiente:

- Opciones de configuración de red:
  - Solo IPv4 y clústeres de pila doble en `dual_stack subnets`.

  - Clústeres solo de IPv6 en subredes solo de IPv6.
- Administración de grupos de subredes:
  - Creación de grupos de subredes compatibles con solo IPv4, solo IPv6 o doble pila
  - Modificación de los grupos de subredes existentes con subredes de VPC adicionales
  - Agregación de subredes solo IPv6 a los grupos de subredes configurados para IPv6
  - Agregación de subredes de IPv4 o de doble pila a grupos configurados de IPv4 o de doble pila
- Configuración de cliente:
  - Al realizar llamadas del plano de datos, puede establecer el protocolo IP preferido para los clústeres de doble_pila mediante:
    - Parámetro `ip_discovery` en Python SDK
    - Parámetro `ipDiscovery` en otros SDK
  - Predeterminado: IPv4 cuando no se ha especificado la preferencia de protocolo

Antes de implementar IPv6 en los clústeres de DAX, debe tener en cuenta lo siguiente:

- No se puede cambiar el tipo de red después de la creación del clúster
- En el caso de los clústeres de doble pila, el parámetro `ip_discovery/ipDiscovery` de la configuración del cliente determina el protocolo IP que se debe utilizar (IPv4 o IPv6)
- Diferentes aplicaciones pueden conectarse al mismo clúster de doble pila mediante distintos protocolos IP en función de su configuración

Example Configuración de cliente de ejemplo

```
DynamoDbAsyncClient client = ClusterDaxAsyncClient.builder()
        .overrideConfiguration(Configuration.builder()
            .url(endpoint)             // DAX cluster endpoint
            .ipDiscovery(ipDiscovery)       // IP discovery type (IPv4 or IPv6)
            .build())
        .build();
```

> **Important**
>
> Cuando utilice políticas de IAM basadas en recursos para restringir las direcciones IP de las tablas de DynamoDB en entornos únicamente de IPv6 con DAX, debe crear una excepción para el rol de IAM del clúster de DAX si bloquea el espacio de direcciones

IPv4 (`0.0.0.0/0`). Agregue una condición `ArnNotEquals` a la política que permita específicamente el acceso al rol de IAM del clúster de DAX y, al mismo tiempo, mantenga las restricciones basadas en IP para otras rutas de acceso. Sin esta excepción, DAX no puede acceder a la tabla de DynamoDB.
Por ejemplo:

Example

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": "*",
      "Action": "dynamodb:PutItem",
      "Resource": "arn:aws:dynamodb:us-
east-1:123456789012:table/MusicCollection",
      "Condition": {
        "ArnNotEquals": {
          "aws:PrincipalArn": "arn:aws:iam::123456789012:role/
DAXServiceRoleForDynamoDBAccess"
        },
        "IpAddress": {
          "aws:SourceIp": "0.0.0.0/0"
        }
      }
    }
  ]
}
```

Creación de un clúster de DAX mediante la AWS CLI

En esta sección se describe cómo crear un clúster de Amazon DynamoDB Accelerator (DAX) mediante la AWS Command Line Interface (AWS CLI). Si aún no lo ha hecho, debe instalar y configurar la AWS CLI. Para hacerlo, vea las siguientes instrucciones en la Guía del usuario de AWS Command Line Interface:

- Instalación del AWS CLI
- Configuración de la AWS CLI

> **Important**
>
> Para administrar los clústeres de DAX con la AWS CLI, instale o actualice a la versión 1.11.110 o superior.

En todos los ejemplos relativos a la AWS CLI se utiliza la región `us-west-2` e identificadores de cuenta ficticios.

Temas

- Paso 1: crear un rol de servicio de IAM para que DAX obtenga acceso a DynamoDB mediante la AWS CLI
- Paso 2: crear un grupo de subredes
- Paso 3: crear un clúster de DAX mediante la AWS CLI
- Paso 4: configurar las reglas de entrada del grupo de seguridad mediante la AWS CLI

Paso 1: crear un rol de servicio de IAM para que DAX obtenga acceso a DynamoDB mediante la AWS CLI

Antes de crear un clúster de Amazon DynamoDB Accelerator (DAX), tiene que crear una función del servicio para él. Una función del servicio es un rol de AWS Identity and Access Management (IAM) que autoriza a un servicio de AWS para actuar en su nombre. La función del servicio permitirá a DAX a acceder a las tablas de DynamoDB como si usted estuviese abriéndolas directamente.

En este paso, se crea una política de IAM y se asocia a un rol de IAM. Esto le permitirá asignar el rol a un clúster de DAX de tal forma que pueda llevar a cabo operaciones de DynamoDB en su nombre.

Para crear un rol de servicio de IAM para DAX

1. Cree un archivo denominado `service-trust-relationship.json` con el siguiente contenido.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
       {
            "Effect": "Allow",
            "Principal": {
                "Service": "dax.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. Cree el rol de servicio.

```
aws iam create-role \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --assume-role-policy-document file://service-trust-relationship.json
```

3. Cree un archivo denominado `service-role-policy.json` con el siguiente contenido.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
```

```
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:111122223333:*"
            ]
        }
    ]
}
```

Sustituya *accountID* por el ID de la cuenta de AWS. Para encontrar su ID de cuenta de AWS, en la esquina superior derecha de la consola, elija su ID de inicio de sesión. El ID de su cuenta de AWS aparece en el menú desplegable.

En el nombre de recurso de Amazon (ARN) en el ejemplo, *accountID* debe ser un número de 12 dígitos. No utilice guiones ni ningún otro signo de puntuación.

4. Cree una política de IAM para la función del servicio.

```
aws iam create-policy \
    --policy-name DAXServicePolicyForDynamoDBAccess \
    --policy-document file://service-role-policy.json
```

En el resultado, tome nota del ARN de la política que ha creado, como en el siguiente ejemplo.

```
arn:aws:iam::123456789012:policy/DAXServicePolicyForDynamoDBAccess
```

5. Asocie la política al rol de servicio. Sustituya *arn* en el siguiente código por el ARN real del rol del paso anterior.

```
aws iam attach-role-policy \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --policy-arn arn
```

A continuación, especifique un grupo de subredes para su VPC predeterminada. Un grupo de subredes es una colección de una o varias subredes de la VPC. Consulte Paso 2: crear un grupo de subredes.

Paso 2: crear un grupo de subredes

Siga este procedimiento para crear un grupo de subred para su clúster de Amazon DynamoDB Accelerator (DAX) mediante la AWS Command Line Interface (AWS CLI).

> **Note**
>
> Si ya ha creado un grupo de subredes para la VPC predeterminada, puede omitir este paso.

DAX está diseñado para ejecutarse dentro de un entorno de Amazon Virtual Private Cloud (Amazon VPC). Si creó la cuenta de AWS después del 4 de diciembre de 2013, ya dispone de una VPC predeterminada en cada región de AWS. Para obtener más información acerca de las VPC predeterminadas, consulte VPC predeterminada y subredes predeterminadas en la Guía del usuario de Amazon VPC.

Para crear un grupo de subredes

1. Para determinar el identificador de la VPC predeterminada, introduzca el siguiente comando.

   ```
   aws ec2 describe-vpcs
   ```

   En el resultado, tome nota del identificador de la VPC predeterminada, como en el siguiente ejemplo.

   ```
   vpc-12345678
   ```

2. Determine los identificadores de las subredes asociadas a la VPC predeterminada. Sustituya *vpcID* por el ID de la VPC real (por ejemplo, `vpc-12345678`).

   ```
   aws ec2 describe-subnets \
       --filters "Name=vpc-id,Values=vpcID" \
       --query "Subnets[*].SubnetId"
   ```

   En la salida, anote los identificadores de subred (por ejemplo,)., `subnet-11111111`.

3. Cree el grupo de subredes. Asegúrese de especificar al menos un ID de subred en el parámetro `--subnet-ids`.

   ```
   aws dax create-subnet-group \
       --subnet-group-name my-subnet-group \
       --subnet-ids subnet-11111111 subnet-22222222 subnet-33333333 subnet-44444444
   ```

Para crear el clúster, consulte Paso 3: crear un clúster de DAX mediante la AWS CLI.

Paso 3: crear un clúster de DAX mediante la AWS CLI

Siga este procedimiento para crear un grupo de subred para su clúster de Amazon DynamoDB Accelerator (DAX) en la Amazon VPC predeterminada mediante la AWS Command Line Interface (AWS CLI).

Para crear un clúster de DAX

1. Obtenga el nombre de recurso de Amazon (ARN) del rol de servicio.

```
aws iam get-role \
    --role-name DAXServiceRoleForDynamoDBAccess \
    --query "Role.Arn" --output text
```

   En el resultado, tome nota del ARN del rol de servicio, como en el siguiente ejemplo.

```
arn:aws:iam::123456789012:role/DAXServiceRoleForDynamoDBAccess
```

2. Crear un clúster de DAX. Sustituya *roleARN* por el ARN del paso anterior.

```
aws dax create-cluster \
    --cluster-name mydaxcluster \
    --node-type dax.r4.large \
    --replication-factor 3 \
    --iam-role-arn roleARN \
    --subnet-group my-subnet-group \
    --sse-specification Enabled=true \
    --region us-west-2
```

   Todos los nodos en el clúster son del tipo `dax.r4.large` (`--node-type`). Hay tres nodos (`--replication-factor`), un nodo primario y dos réplicas.

> **Note**
>
> Ya que `sudo` y `grep` son palabras clave reservadas, no puede crear un clúster DAX con estas palabras en el nombre del clúster. Por ejemplo, `sudo` y `sudocluster` son nombres de clúster no válidos.

Para ver el estado del clúster, introduzca el siguiente comando.

```
aws dax describe-clusters
```

El estado se muestra en el resultado (por ejemplo:)., `"Status": "creating"`.

> **Note**
>
> La creación del clúster tarda varios minutos. Cuando el clúster esté listo, su estado cambiará a `available`. Mientras tanto, continúe con Paso 4: configurar las reglas de entrada del grupo de seguridad mediante la AWS CLI y siga las instrucciones que se facilitan.

Paso 4: configurar las reglas de entrada del grupo de seguridad mediante la AWS CLI

Los nodos del clúster del Amazon DynamoDB Accelerator (DAX) utilizan el grupo de seguridad predeterminado para su Amazon VPC. Para el grupo de seguridad predeterminado, debe autorizar el tráfico entrante en el puerto TCP 8111 para clústeres no cifrados o en el puerto 9111 para clústeres cifrados. Esto permitirá que las instancias de Amazon EC2 de la Amazon VPC; accedan al clúster de DAX.

> **Note**
>
> Si ha lanzado el clúster de DAX con un grupo de seguridad distinto (que no sea `default`), tendrá que llevar a cabo este procedimiento para ese grupo.

Para configurar las reglas de entrada del grupo de seguridad

1. Para determinar el identificador del grupo de seguridad predeterminado, introduzca el siguiente comando. Sustituya *`vpcID`* por el ID de la VPC real (de Paso 2: crear un grupo de subredes).

   ```
   aws ec2 describe-security-groups \
       --filters Name=vpc-id,Values=vpcID Name=group-name,Values=default \
       --query "SecurityGroups[*].{GroupName:GroupName,GroupId:GroupId}"
   ```

   En el resultado, tome nota del identificador del grupo de seguridad (por ejemplo,)., `sg-01234567`.
2. A continuación, introduzca lo siguiente. Sustituya *`sgID`* por el identificador del grupo de seguridad real. Use el puerto 8111 para clústeres no cifrados y el 9111 para clústeres cifrados.

```
aws ec2 authorize-security-group-ingress \
    --group-id sgID --protocol tcp --port 8111
```

Creación de un clúster de DAX mediante la Consola de administración de AWS

Esta sección describe cómo crear un clúster de Amazon DynamoDB Accelerator (DAX) mediante la Consola de administración de AWS.

Temas

- Paso 1: crear un grupo de subred mediante la Consola de administración de AWS
- Paso 2: crear un clúster de DAX mediante la Consola de administración de AWS
- Paso 3: configurar las reglas de entrada del grupo de seguridad mediante la Consola de administración de AWS

Paso 1: crear un grupo de subred mediante la Consola de administración de AWS

Siga este procedimiento para crear un grupo de subred para su clúster de Amazon DynamoDB Accelerator (DAX) mediante la Consola de administración de AWS.

> **Note**
>
> Si ya ha creado un grupo de subredes para la VPC predeterminada, puede omitir este paso.

DAX está diseñado para ejecutarse dentro de un entorno de Amazon Virtual Private Cloud (Amazon VPC). Si creó la cuenta de AWS después del 4 de diciembre de 2013, ya dispone de una VPC predeterminada en cada región de AWS. Para obtener más información acerca de las VPC predeterminadas, consulte VPC predeterminada y subredes predeterminadas en la Guía del usuario de Amazon VPC.

Como parte del proceso de creación de un clúster de DAX, debe especificar un grupo de subredes. Un grupo de subredes es una colección de una o varias subredes de la VPC. Al crear un clúster de DAX, los nodos se implementan en las subredes del grupo de subredes.

> **Note**
>
> La VPC que tiene este clúster de DAX puede contener otros recursos e incluso puntos de conexión de VPC para los demás servicios, excepto el punto de conexión de VPC para ElastiCache y puede provocar errores en las operaciones del clúster de DAX.

Para crear un grupo de subredes

1. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación, en DAX, elija Subnet groups (Grupo de subredes).
3. Elija Create subnet group (Crear grupo de subredes).
4. En la ventana Create subnet group (Crear grupo de subredes), haga lo siguiente:
   a. Name (Nombre): ingrese un nombre corto para el grupo de subredes.
   b. Description (Descripción): ingrese una descripción del grupo de subredes.
   c. VPC ID (ID de la VPC): elija el identificador del entorno de Amazon VPC.
   d. Subnets (Subred): elija una o varias subredes de la lista.

   > **Note**
   >
   > Las subredes se distribuyen entre varias zonas de disponibilidad. Si tiene previsto crear un clúster de DAX de varios nodos (un nodo primario y una o más réplicas de lectura), recomendamos elegir varios identificadores de subredes. A continuación, DAX puede implementar los nodos del clúster en varias zonas de disponibilidad. Si una zona de disponibilidad deja de estar disponible, DAX realiza automáticamente una conmutación por error a una de las zonas de disponibilidad restantes. El clúster de DAX seguirá funcionando sin interrupción.

   Cuando esté conforme con los ajustes, elija Create subnet group (Crear grupo de subredes).

Para crear el clúster, consulte Paso 2: crear un clúster de DAX mediante la Consola de administración de AWS.

Paso 2: crear un clúster de DAX mediante la Consola de administración de AWS

Siga este procedimiento para crear un clúster de Amazon DynamoDB Accelerator (DAX) en la Amazon VPC predeterminada.

Para crear un clúster de DAX

1. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación, en DAX, elija Clusters (Clústeres).
3. Elija Create cluster.
4. En la ventana Create cluster (Crear clúster), haga lo siguiente:

    a. Cluster name (Nombre del clúster): ingrese un nombre corto para su clúster de DAX.

    > **Note**
    >
    > Ya que `sudo` y `grep` son palabras clave reservadas, no puede crear un clúster DAX con estas palabras en el nombre del clúster. Por ejemplo, `sudo` y `sudocluster` son nombres de clúster no válidos.

    b. Cluster description (Descripción del clúster): ingrese una descripción del clúster.

    c. Node types (Tipos de nodo): elija el tipo de nodo de todos los nodos del clúster.

    d. Cluster size (Tamaño del clúster): elija la cantidad de nodos del clúster. Un clúster consta de un nodo principal y hasta nueve réplicas de lectura.

    > **Note**
    >
    > Si desea crear un clúster de un solo nodo, elija 1. Su clúster constará de un nodo principal.
    > Si desea crear un clúster de varios nodos, elija un número entre 3 (una instancia principal y dos réplicas de lectura) y 10 (una instancia principal y nueve réplicas de lectura).
    >
    > > **Important**
    > >
    > > Para su uso en producción, recomendamos encarecidamente utilizar DAX con al menos tres nodos, en donde cada nodo se coloca en zonas de

disponibilidad diferentes. Se requieren tres nodos para que un clúster de DAX sea tolerante a errores.
Un clúster de DAX se puede implementar con uno o dos nodos para cargas de trabajo de desarrollo o pruebas. Los clústeres de uno o dos nodos no son tolerantes a errores y no se recomiendan menos de tres nodos para su uso en producción. Si los clústeres de uno o dos nodos tienen errores de software o hardware, el clúster podría dejar de estar disponible o podrían perderse los datos almacenados en caché.

e. Elija Siguiente.

f. Subnet group (Grupo de subred): seleccione Choose existing (Elegir existente) y elija el grupo de subred que creó en Paso 1: crear un grupo de subred mediante la Consola de administración de AWS.

g. Access control (Control de acceso): elija el grupo de seguridad default (predeterminado).

h. Zonas de disponibilidad (AZ): elija Automatic (Automático).

i. Elija next (siguiente).

j. Función del servicio de IAM para acceder a DynamoDB: elija Create new (Crear nuevo) e ingrese la siguiente información:

- IAM role Name (Nombre del rol de IAM): ingrese un nombre para el rol de IAM, (por ejemplo, `DAXServiceRole`). La consola creará un nuevo rol de IAM y el clúster de DAX lo asumirá en el tiempo de ejecución.
- Seleccione la casilla situada junto a Create policy (Crear una política).
- IAM role policy (Política del rol de IAM): elija Read/Write (Lectura/escritura). Esto permite que el clúster de DAX lleve a cabo operaciones de lectura y escritura en DynamoDB.
- Nombre de la nueva política de IAM: este campo se irá rellenando a medida que introduzca el nombre del rol de IAM. También puede introducir un nombre para la política de IAM, por ejemplo, `DAXServicePolicy`. La consola creará una nueva política de IAM y se la adjuntará al rol de IAM.
- Acceso a tablas de DynamoDB: elija Todas las tablas.

k. Cifrado: elija Activar el cifrado en reposo y Activar el cifrado en tránsito. Para obtener más información, consulte Cifrado en reposo de DAX y Cifrado en tránsito de DAX.

También se requiere una función del servicio independiente para que DAX acceda a Amazon EC2. DAX le crea automáticamente esta función del servicio. Para obtener más información, consulte Uso de roles vinculados a servicios para DAX.

5. Cuando esté conforme con la configuración, elija Siguiente.
6. Grupo de parámetros: elija Elegir existente.
7. Periodo de mantenimiento: elija Sin preferencia si no tiene preferencia cuando se aplican las actualizaciones de software, o elija Especificar periodo de tiempo y proporcione las opciones Día de la semana, Hora (UTC) y Comienza dentro de (horas) para programar su intervalo de mantenimiento.
8. Etiquetas: elija Agregar nueva etiqueta para introducir un par clave/valor con fines de etiquetado.
9. Elija Siguiente.

En la pantalla Revisar y crear, puede revisar todas las configuraciones. Si está listo para crear el clúster, elija Crear clúster.

En la pantalla Clusters (Clústeres), aparecerá el clúster de DAX con el estado Creating (En creación).

> **Note**
>
> La creación del clúster tarda varios minutos. Cuando el clúster esté listo, su estado cambiará a Available (Disponible).
> Mientras tanto, continúe con Paso 3: configurar las reglas de entrada del grupo de seguridad mediante la Consola de administración de AWS y siga las instrucciones que se facilitan.

Paso 3: configurar las reglas de entrada del grupo de seguridad mediante la Consola de administración de AWS

El clúster de Amazon DynamoDB Accelerator (DAX) se comunica a través del puerto TCP 8111 (para clústeres no cifrados) o 9111 (para clústeres cifrados), así que debe autorizar el tráfico entrante en ese puerto. Esto permitirá que las instancias de Amazon EC2 de la Amazon VPC; accedan al clúster de DAX.

> **Note**
>
> Si ha lanzado el clúster de DAX con un grupo de seguridad distinto (que no sea `default`), tendrá que llevar a cabo este procedimiento para ese grupo.

Para configurar las reglas de entrada del grupo de seguridad

1. Abra la consola de Amazon EC2 en https://console.aws.amazon.com/ec2/.
2. En el panel de navegación, elija Grupos de seguridad.
3. Elija el grupo de seguridad default (predeterminada). En el menú Actions (Acciones), elija Edit inbound rules (Editar reglas de entrada).
4. Elija Add Rule (Añadir regla) y especifique la siguiente información:
   - Port Range (Rango de puertos): ingrese 8111 (si su clúster no está cifrado) o 9111 (si su clúster está cifrado).
   - Origen: déjelo como Personalizado y elija el campo de búsqueda de la derecha. Se mostrará un menú desplegable. Elija el identificador del grupo de seguridad predeterminado.
5. Elija Guardar para guardar los cambios.
6. Para actualizar el nombre en la consola, vaya a la propiedad Nombre y elija la opción Editar que aparece.

Modelos de coherencia de DAX y DynamoDB

Amazon DynamoDB Accelerator (DAX) es un servicio de almacenamiento en caché de escritura indirecta (write-through), diseñado para simplificar el proceso de agregar una caché a las tablas de DynamoDB. Dado que DAX funciona por separado de DynamoDB, es importante comprender los modelos de consistencia de DAX y de DynamoDB para asegurarse de que sus aplicaciones se comporten como se espera de ellas.

En muchos casos de uso, la forma de usar DAX en la aplicación afecta a la consistencia de datos en el clúster de DAX, así como a la consistencia de datos entre DAX y DynamoDB.

Temas

- Coherencia entre los nodos de los clústeres de DAX
- Comportamiento de la caché de elementos de DAX

- Comportamiento de la caché de consulta de DAX
- Lecturas altamente coherentes y transaccionales
- Almacenamiento en caché negativo
- Estrategias de escritura

Coherencia entre los nodos de los clústeres de DAX

Para dotar de alta disponibilidad a la aplicación, recomendamos aprovisionar el clúster de DAX con al menos tres nodos. A continuación, coloque los nodos en varias zonas de disponibilidad de una región.

Mientras el clúster de DAX se está ejecutando, replica los datos entre todos los nodos del clúster (siempre y cuando haya aprovisionado más de un nodo). Tomemos una aplicación que realiza correctamente una operación `UpdateItem` mediante DAX. Esta acción hace que la caché de elementos del nodo principal se modifique con el nuevo valor. A continuación, este valor se replica en todos los demás nodos del clúster. Esta replicación presenta consistencia final y suele tardar menos de un segundo en llevarse a cabo.

En esta situación, es posible que dos clientes lean la misma clave del mismo clúster de DAX pero reciban valores distintos, según a qué nodo haya obtenido acceso cada cliente. Todos los nodos serán consistentes cuando la actualización haya terminado de replicarse en todos los nodos del clúster. (Este comportamiento se asemeja a la naturaleza eventualmente consistente de DynamoDB).

Si va a crear una aplicación que utiliza DAX, dicha aplicación debe diseñarse de forma que tolere los datos de consistencia final.

Comportamiento de la caché de elementos de DAX

Cada clúster de DAX presenta dos memorias caché diferentes, una caché de elemento y una caché de consulta. Para obtener más información, consulte DAX: cómo funciona.

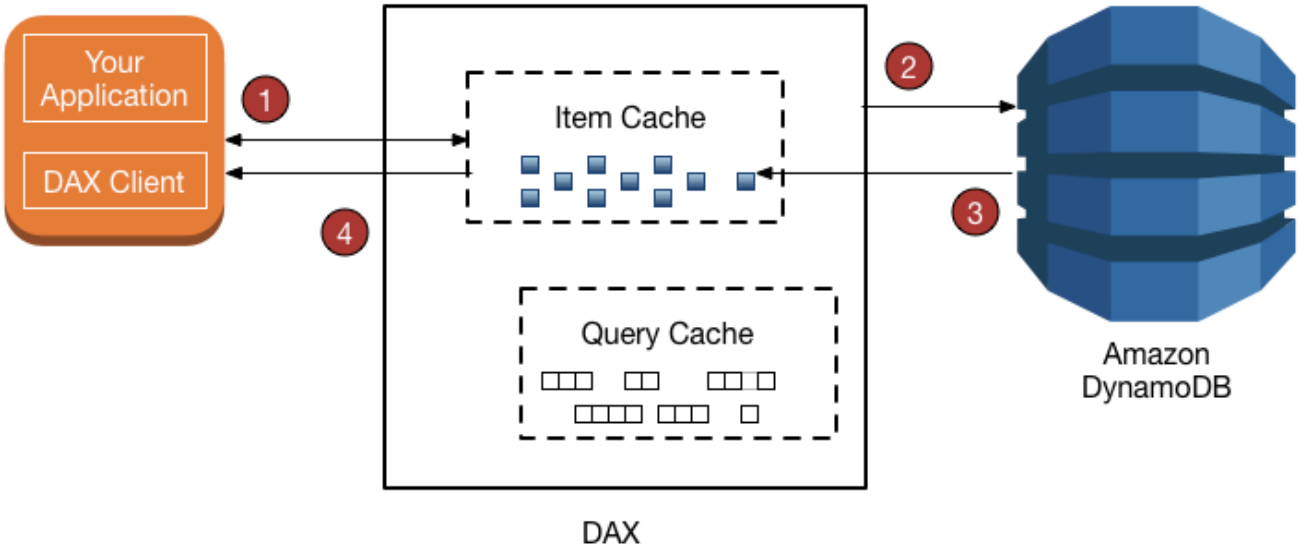
En esta sección se abordan las implicaciones para la consistencia de la lectura y la escritura en la caché de elemento de DAX.

Coherencia de lectura

De forma predeterminada, con DynamoDB la operación `GetItem` lleva a cabo una lectura eventualmente consistente. Suponga que utiliza `UpdateItem` con el cliente de DynamoDB. Si

inmediatamente después intenta leer el mismo elemento, es posible que los datos aparezcan igual que antes de la actualización. Esto se debe al retraso de propagación entre todas las ubicaciones de almacenamiento de DynamoDB. Normalmente, se logra la coherencia en cuestión de segundos. Si reintenta la lectura, probablemente verá el elemento actualizado.

Cuando utiliza `GetItem` con el cliente de DAX, la operación (en este caso, una lectura eventualmente consistente) se lleva a cabo como se muestra a continuación.



1. El cliente de DAX emite una solicitud `GetItem`. DAX intenta leer el elemento solicitado en la caché de elemento. Si el elemento está presente en la caché (acierto de caché), DAX se lo devuelve a la aplicación.
2. Si el elemento no está disponible (error de caché), DAX realiza una operación `GetItem` de consistencia final en DynamoDB.
3. DynamoDB devuelve el elemento solicitado y DAX lo almacena en la caché de elemento.
4. DAX devuelve el elemento a la aplicación.
5. (No se muestra) Si el clúster de DAX contiene más de un nodo, el elemento se replica en todos los demás nodos del clúster.

El elemento permanecerá en la caché de elemento de DAX, sujeto a la configuración del período de vida (TTL) y el algoritmo de elementos menos usados recientemente (LRU) de la caché. Para obtener más información, consulte DAX: cómo funciona.

Sin embargo, durante este periodo, DAX no vuelve a leer el elemento en DynamoDB. Si otra persona actualiza el elemento mediante un cliente de DynamoDB eludiendo por completo a DAX, entonces una solicitud `GetItem` realizada con el cliente de DAX dará un resultado distinto que si esa misma solicitud `GetItem` se lleva a cabo mediante el cliente de DynamoDB. Si esto sucede, DAX y DynamoDB contendrán valores inconsistentes para la misma clave hasta que venza el TTL del elemento de DAX.

Si una aplicación modifica los datos de una tabla de DynamoDB subyacente eludiendo DAX, la aplicación tendrá que prever y tolerar las inconsistencias de datos que surjan.

> **Note**
>
> Además de `GetItem`, el cliente de DAX también admite solicitudes `BatchGetItem`. En esencia, `BatchGetItem` es un encapsulador que contiene una o varias solicitudes `GetItem`, de modo que DAX trata cada una de ellas como una operación `GetItem` individual.

Coherencia de escritura

DAX es una caché de escritura indirecta (write-through). Esto simplifica el proceso de mantener la consistencia entre la caché de elemento de DAX y las tablas de DynamoDB subyacentes.

El cliente de DAX admite las mismas operaciones de API de escritura que DynamoDB (`PutItem`, `UpdateItem`, `DeleteItem`, `BatchWriteItem` y `TransactWriteItems`). Cuando se utilizan estas operaciones con el cliente de DAX, los elementos se modifican tanto en DAX como en DynamoDB. DAX actualizará los elementos en la caché de elemento, independientemente del valor de TTL que tengan.

Por ejemplo, supongamos que emite una solicitud `GetItem` del cliente de DAX para leer un elemento de la tabla `ProductCatalog`. (La clave de partición es `Id` y no hay clave de ordenación). Recupera el elemento cuyo `Id` es `101`. El valor `QuantityOnHand` para ese elemento es `42`. DAX almacena el elemento en su caché de elemento con un TTL concreto. En este ejemplo, vamos a suponer que el TTL es de diez minutos. A continuación, 3 minutos más tarde, otra aplicación utiliza

el cliente de DAX para actualizar el mismo elemento, de forma que su valor de `QuantityOnHand` pasa a ser `41`. Suponiendo que el elemento no se vuelva a actualizar, todas las lecturas posteriores del mismo elemento que tengan lugar en los próximos diez minutos devolverán el valor de `QuantityOnHand` almacenado en caché (`41`).

Cómo procesa DAX las escrituras

DAX está pensado para aplicaciones que requieren lecturas de alto rendimiento. Como caché de escritura indirecta, DAX pasa las escrituras a DynamoDB de forma sincrónica y, a continuación, replica de forma automática y asíncrona las actualizaciones resultantes a la caché de elemento en todos los nodos del clúster. No es necesario administrar la lógica de invalidación de caché, porque DAX lo hace automáticamente.

DAX admite las siguientes operaciones de escritura: `PutItem`, `UpdateItem`, `DeleteItem`, `BatchWriteItem` y `TransactWriteItems`.

Cuando se envía una solicitud `PutItem`, `UpdateItem`, `DeleteItem` o `BatchWriteItem` a DAX, ocurre lo siguiente:

- DAX envía la solicitud a DynamoDB.
- DynamoDB responde a DAX para confirmar que la escritura se ha llevado a cabo correctamente.
- DAX escribe el elemento en su caché de elemento.
- DAX indica al solicitante que la escritura se ha realizado correctamente.

Cuando se envía una solicitud `TransactWriteItems` a DAX, ocurre lo siguiente:

- DAX envía la solicitud a DynamoDB.
- DynamoDB responde a DAX para confirmar que la transacción se ha completado.
- DAX indica al solicitante que la escritura se ha realizado correctamente.
- En segundo plano, DAX realiza una solicitud `TransactGetItems` para que cada elemento en la solicitud `TransactWriteItems` almacene el elemento en la caché de elemento. `TransactGetItems` se utiliza para garantizar un aislamiento serializable.

Si una escritura no se puede realizar en DynamoDB por cualquier motivo, incluida la limitación controlada, el elemento no se almacenará en caché en DAX. Se devuelve al solicitante la excepción correspondiente al error. De este modo se garantiza que los datos no se escriban en la caché de DAX a no ser que antes se hayan escrito correctamente en DynamoDB.

> **Note**
>
> Cada escritura en DAX altera el estado de la caché de elemento. Sin embargo, las escrituras en ella no afectan a la caché de consultas. (La caché de elemento y la caché de consulta de DAX se utilizan con fines diferentes y operan de manera independiente una de la otra).

Comportamiento de la caché de consulta de DAX

DAX almacena en su caché de consulta los resultados de las solicitudes `Query` y `Scan`. Sin embargo, estos resultados no afectan en absoluto a la caché de elementos. Cuando la aplicación emite una solicitud `Query` o `Scan` con DAX, el conjunto de resultados se guarda en la caché de consulta, no en la de elemento. No se puede "calentar" la caché de elementos antes de llevar a cabo una operación `Scan`, ya que la caché de elementos y la caché de consultas son entidades independientes.

Coherencia de consulta-actualización-consulta

Las actualizaciones de la caché de elementos o de la tabla de DynamoDB subyacente no invalidan ni modifican los resultados almacenados en la caché de consultas.

A modo de ejemplo, considere la siguiente situación. Una aplicación está trabajando con la tabla `DocumentRevisions`, que tiene una clave de partición `DocId` y una clave de ordenación `RevisionNumber`.

1. Un cliente emite una `Query` para `DocId 101`, para todos los elementos con `RevisionNumber` mayor o igual que `5`. DAX almacena el conjunto de resultados en la caché de consulta y se lo devuelve al usuario.
2. El cliente emite una solicitud `PutItem` para `DocId 101` con un valor de `RevisionNumber` de `20`.
3. El cliente emite la misma operación `Query` que se describe en el paso 1 (`DocId 101` y `RevisionNumber >= 5`).

En este caso, el conjunto de resultados almacenado en caché correspondiente a la operación `Query` emitida en el paso 3 es idéntico al que se almacenó en caché en el paso 1. El motivo es que DAX no invalida los conjuntos de resultados de `Query` o `Scan` cuando se actualizan elementos individuales. La operación `PutItem` del paso 2 solo se refleja en la caché de consulta de DAX cuando vence el TTL de la `Query`.

La aplicación debe tener en cuenta el valor de TTL de la caché de consultas y el tiempo durante el cual la aplicación puede tolerar resultados inconsistentes entre las cachés de consultas y de elementos.

Lecturas altamente coherentes y transaccionales

Para realizar una solicitud de lectura fuertemente consistente `GetItem`, `BatchGetItem`, `Query` o `Scan`, debe establecer el parámetro `ConsistentRead` en true (verdadero). DAX pasa las solicitudes de lectura fuertemente consistentes a DynamoDB. Cuando recibe una respuesta de DynamoDB, DAX devuelve los resultados al cliente, pero no almacena en caché los resultados. DAX no puede atender por sí solo a las lecturas fuertemente consistentes, porque no está estrechamente acoplado a DynamoDB. Debido a esto, todas las lecturas posteriores de DAX tendrían que ser lecturas eventualmente consistentes. Y las lecturas fuertemente consistentes posteriores tendrían que pasarse a través de DynamoDB.

DAX administra las solicitudes `TransactGetItems` de la misma manera que administra las lecturas fuertemente consistentes. DAX pasa todas las solicitudes `TransactGetItems` a DynamoDB. Cuando recibe una respuesta de DynamoDB, DAX devuelve los resultados al cliente, pero no almacena en caché los resultados.

Almacenamiento en caché negativo

DAX admite las entradas de caché negativas, tanto en la caché de elemento como en la caché de consulta. Una entrada de caché negativa se produce cuando DAX no encuentra los elementos solicitados en una tabla de DynamoDB subyacente. En lugar de generar un error, DAX almacena en caché un resultado vacío y se lo devuelve al usuario.

Por ejemplo, supongamos que una aplicación envía una solicitud `GetItem` a un clúster de DAX, pero que no hay ningún elemento coincidente en la caché de elemento de DAX. Esto hace que DAX lea el elemento correspondiente en la tabla de DynamoDB subyacente. Si el elemento tampoco está presente en DynamoDB, entonces DAX almacena un elemento vacío en su caché de elemento y, a continuación, devuelve este elemento vacío a la aplicación. Ahora, supongamos que la aplicación envía otra solicitud `GetItem` para obtener el mismo elemento. DAX encontrará el elemento vacío en la caché de elemento y se lo devolverá de inmediato a la aplicación. No consultará DynamoDB en absoluto.

Una entrada de caché negativa se conserva en la caché de elemento de DAX hasta que vence el TTL del elemento, se invoca su LRU o el elemento se modifica mediante `PutItem`, `UpdateItem` o `DeleteItem`.

La caché de consulta de DAX administra los resultados de caché negativos de forma similar. Si una aplicación realiza una operación `Query` o `Scan` y la caché de consulta de DAX no contiene un resultado en caché, DAX envía la solicitud a DynamoDB. Si no hay elementos coincidentes en el conjunto de resultados, DAX almacena un conjunto de resultados vacío en la caché de consulta y devuelve el conjunto de resultados vacío a la aplicación. Las solicitudes `Query` o `Scan` producen el mismo conjunto de resultados (vacío), hasta que vence su TTL.

Estrategias de escritura

El comportamiento de escritura indirecta (write-through) de DAX es adecuado para muchos patrones de aplicaciones. No obstante, existen algunos patrones de aplicación en los que este modelo de escritura indirecta podría no ser el adecuado.

En las aplicaciones que son sensibles a la latencia, la escritura a través de DAX genera un salto de red más. Por eso, la escritura en DAX es algo más lenta que si se realiza directamente en DynamoDB. Si la aplicación es sensible a la latencia de escritura, puede reducir esa latencia escribiendo directamente en DynamoDB. Para obtener más información, consulte Escritura directa.

Para las aplicaciones con gran intensidad de escritura (como las que llevan a cabo la carga de datos masivos), no es deseable escribir todos los datos a través de DAX, puesto que la aplicación solo leerá un pequeño porcentaje de ellos. Cuando se escriben grandes cantidades de datos a través de DAX, debe invocar el algoritmo LRU con el fin de dejar espacio en la caché para los nuevos elementos que hay que leer. Esto reduce la eficacia de DAX como caché de lectura.

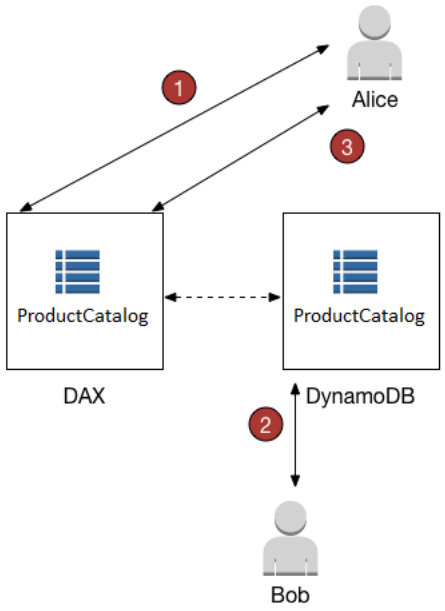
Al escribir un elemento en DAX, el estado de la caché de elemento se modifica para dar cabida al nuevo elemento. (Por ejemplo, DAX podría tener que expulsar datos antiguos de la caché de elemento con el fin de dejar espacio para el nuevo elemento). El nuevo elemento permanecerá en la caché de elementos, de conformidad con el algoritmo LRU y el ajuste de TTL de esta última. Mientras el elemento persista en la caché de elemento, DAX no volverá a leerlo en DynamoDB.

Escritura indirecta

La caché de elemento de DAX implementa una política de escritura indirecta. Para obtener más información, consulte Cómo procesa DAX las escrituras.

Cuando se escribe un elemento, DAX se asegura de que el elemento almacenado en caché esté sincronizado con el elemento presente en DynamoDB. Esto resulta útil para las aplicaciones que tienen que volver a leer un elemento inmediatamente después de escribirlo. Sin embargo, si otras aplicaciones escriben directamente en una tabla de DynamoDB, el elemento contenido en la caché de elemento de DAX dejará de estar sincronizada con DynamoDB.

Por ejemplo, tomemos dos usuarios (Alice y Bob) que están utilizando la tabla `ProductCatalog`. Alice obtiene acceso a la tabla mediante DAX, pero Bob elude DAX y obtiene acceso a la tabla directamente en DynamoDB.



1. Alice actualiza un elemento de la tabla `ProductCatalog`. DAX reenvía la solicitud a DynamoDB y la actualización se lleva a cabo correctamente. Después, DAX escribe el elemento en su caché de elemento y devuelve una respuesta correcta a Alice. A partir de ese momento, hasta que el elemento se expulse definitivamente de la caché, cualquier usuario que lo lea en DAX ve el elemento con la actualización de Alice.
2. Poco después, Bob actualiza el mismo elemento `ProductCatalog` en el que Alice ha escrito. Sin embargo, Bob actualiza el elemento directamente en DynamoDB. DAX no actualiza automáticamente su caché de elemento en respuesta a las actualizaciones a través de DynamoDB. Por tanto, los usuarios de DAX no verán la actualización de Bob.
3. Alice vuelve a leer el elemento en DAX. Este elemento se encuentra en la caché del elemento, por lo que DAX se lo devuelve a Alice sin obtener acceso a la tabla de DynamoDB.

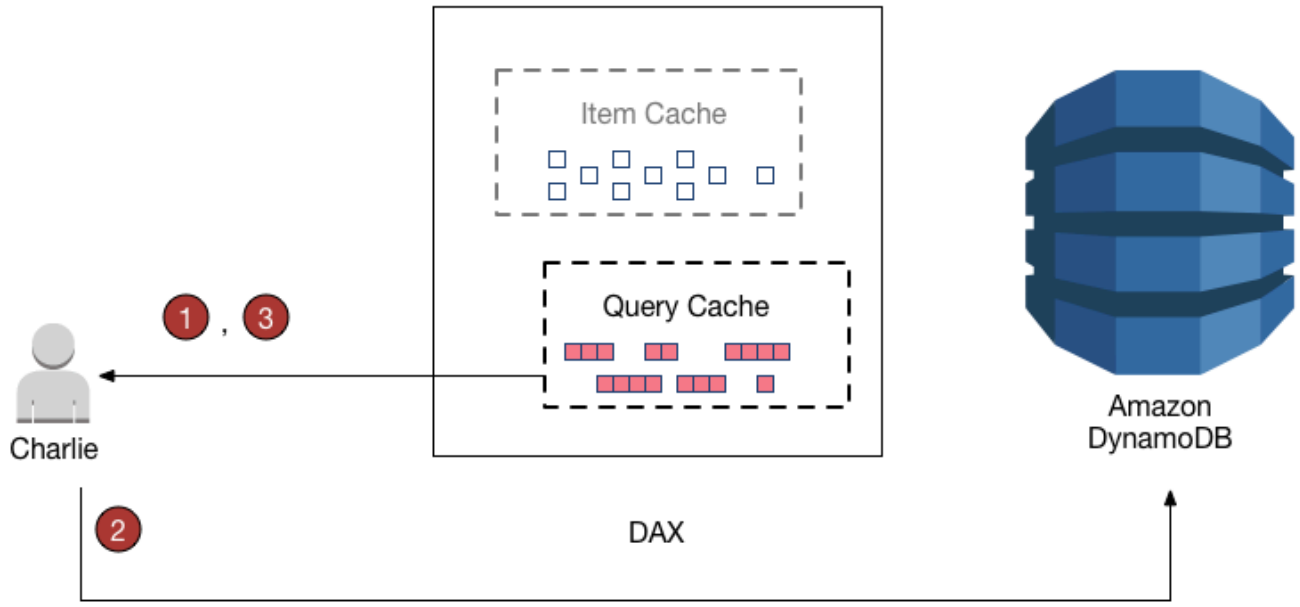
En este caso, Alice y Bob obtienen representaciones distintas del mismo elemento `ProductCatalog`. Esto continuará siendo así hasta que DAX expulse el elemento de la caché de elemento o hasta que otro usuario lo vuelva a actualizar mediante DAX.

Escritura directa

Si la aplicación tiene que escribir grandes cantidades de datos (por ejemplo, en una carga de datos masivos), puede ser más lógico eludir DAX y escribir los datos directamente en DynamoDB. Esta estrategia de escritura directa (write-around) reduce la latencia. Sin embargo, la caché de elemento no se mantendrá sincronizada con los datos de DynamoDB.

Si decide utilizar una estrategia de escritura directa, recuerde que DAX rellena su caché de elemento cada vez que las aplicaciones utilizan el cliente de DAX para leer los datos. Esto puede ser beneficioso en algunos casos, ya que garantiza que solo se almacenen en caché los datos que se leen con mayor frecuencia (en lugar de los datos que se escriben con mayor frecuencia).

Suponga, por ejemplo, que un usuario (Charlie) desea trabajar con una tabla diferente, la tabla `GameScores`, mediante DAX. La clave de partición de `GameScores` es `UserId`, por lo que todas las puntuaciones de Charlie tendrían el mismo `UserId`.



1. Charlie desea recuperar todas sus puntuaciones, por lo que envía una `Query` a DAX. Suponiendo que esta consulta no se haya emitido antes, DAX se la reenvía a DynamoDB para procesarla. Almacena los resultados en la caché de consulta de DAX y, por último, devuelve los resultados a Charlie. El conjunto de resultados seguirá disponible en la caché de consultas hasta que se expulse.

2. Ahora, supongamos que Charlie juega a Blasters Meteor y logra una puntuación máxima. Charlie envía una solicitud `UpdateItem` a DynamoDB para modificar un elemento en la tabla `GameScores`.
3. Por último, Charlie decide volver a ejecutar la operación `Query` anterior para recuperar todos sus datos de `GameScores`. Charlie no ve su puntuación máxima en Meteor Blasters en los resultados. Esto se debe a que los resultados de la consulta proceden de la caché de consultas, no de la caché de elementos. Las dos cachés son independientes entre sí, de modo que un cambio en una de ellas no afecta a la otra.

DAX no actualiza los conjuntos de resultados de la caché de consulta con los datos más recientes de DynamoDB. Cada conjunto de resultados de la caché de consultas estaba actualizado en el momento de ejecutar la operación `Query` o `Scan`. Por lo tanto, los resultados que Charlie obtiene con la operación `Query` no reflejan su operación `PutItem`. Esto seguirá siendo así hasta que DAX expulse el conjunto de resultados de la caché de consulta.

Desarrollo con el cliente de DynamoDB Accelerator (DAX)

Para usar DAX desde una aplicación, use el cliente de DAX para su lenguaje de programación. El cliente de DAX se ha diseñado de tal forma que se minimicen las interrupciones para las aplicaciones de Amazon DynamoDB existentes; solo tiene que realizar algunas modificaciones sencillas en el código.

> **Note**
>
> Los clientes de DAX para diversos lenguajes de programación están disponibles en el siguiente sitio:
>
> - http://dax-sdk.s3-website-us-west-2.amazonaws.com

En esta sección se muestra cómo lanzar una instancia de Amazon EC2 en la Amazon VPC predeterminada, conectarse a la instancia y ejecutar una aplicación de ejemplo. También se ofrece información sobre cómo modificar la aplicación existente para que pueda utilizar el clúster de DAX.

Temas

- Tutorial: Ejecución de un ejemplo de aplicación mediante DynamoDB Accelerator (DAX)
- Modificación de una aplicación existente para que use DAX

Tutorial: Ejecución de un ejemplo de aplicación mediante DynamoDB Accelerator (DAX)

En este tutorial se muestra cómo lanzar una instancia de Amazon EC2 en su nube virtual privada (VPC) predeterminada, conectarse a la instancia y ejecutar una aplicación de ejemplo que utiliza Amazon DynamoDB Accelerator (DAX).

> **Note**
>
> Para completar este tutorial, debe disponer de un clúster de DAX en ejecución en la VPC predeterminada. Si no ha creado un clúster de DAX, consulte Creación de un clúster de DAX para obtener instrucciones.

Temas

- Paso 1: lanzar una instancia de Amazon EC2
- Paso 2: Crear un usuario y una política
- Paso 3: configurar una instancia de Amazon EC2
- Paso 4: ejecutar un ejemplo de aplicación

Paso 1: lanzar una instancia de Amazon EC2

Cuando el clúster del Acelerador de Amazon DynamoDB (DAX) de Amazon esté disponible, podrá iniciar una instancia de Amazon EC2 en la Amazon Virtual Private Cloud (Amazon VPC) predeterminada. Luego, puede instalar y ejecutar el software de cliente de DAX en dicha instancia.

Para iniciar una instancia de EC2

1. Inicie sesión en la Consola de administración de AWS y abra la consola de Amazon EC2 en https://console.aws.amazon.com/ec2/.
2. Elija Launch Instance (Lanzar instancia) y proceda del modo siguiente:

   Paso 1: Elegir una imagen de máquina de Amazon (AMI)

   1. En la lista de AMI, busque la Amazon Linux AMI (AMI de Amazon Linux) y elija Select (Seleccionar).

Paso 2: Elegir un tipo de instancia

1. En la lista de tipos de instancias, elija t2.micro.
2. Elija Next: Configure Instance Details.

Paso 3: Configurar los detalles de la instancia

1. En Network (Red), seleccione su VPC predeterminada.
2. Elija Siguiente: Añadir almacenamiento.

Paso 4: Agregar almacenamiento

1. Omita este paso; para ello, elija Next: Add tags (Siguiente: Añadir etiquetas).

Paso 5: Añadir etiquetas

1. Elija Next: Configure Security Group para omitir este paso.

Paso 6: Configurar un grupo de seguridad

1. Elija Seleccionar un grupo de seguridad existente.
2. En la lista de grupos de seguridad, elija default. Se trata del grupo de seguridad predeterminado para la VPC.
3. Elija Next: Review and Launch.

Paso 7: Revisar el lanzamiento de la instancia

1. Elija Iniciar.

3. En la ventana Select an existing key pair or create a new key pair, proceda del modo siguiente:

   - Si no dispone de un par de claves de Amazon EC2, elija Create a new key pair (Crear un nuevo par de claves) y siga las instrucciones. Se le pedirá que descargue un archivo de claves privadas (archivo `.pem`). Necesitará este archivo más tarde cuando se registre en su instancia de Amazon EC2.

- Si ya dispone de un par de claves de Amazon EC2, vaya a Select a key pair (Seleccione un par de claves) y elija su par de claves de la lista. Debe tener el archivo de clave privada (archivo `.pem`) disponible para poder registrarse en la instancia de Amazon EC2.

4. Después de configurar el par de claves, elija Launch instances (Lanzar instancias).
5. En el panel de navegación de la consola, elija EC2 Dashboard (Panel de EC2) y, a continuación, seleccione la instancia que ha lanzado. En el panel inferior, en la pestaña Description (Descripción), busque el Public DNS (DNS público) de su instancia, por ejemplo: `ec2-11-22-33-44.us-west-2.compute.amazonaws.com`. Anote el nombre de DNS público, pues lo necesita para Paso 3: configurar una instancia de Amazon EC2.

> **Note**
>
> La instancia de Amazon EC2 tardará unos minutos en estar disponible. Mientras tanto, continúe con Paso 2: Crear un usuario y una política y siga las instrucciones que se facilitan.

Paso 2: Crear un usuario y una política

En este paso, creará un usuario con una política que concede acceso al clúster de Acelerador de Amazon DynamoDB (DAX) y a DynamoDB mediante AWS Identity and Access Management. A continuación, puede ejecutar aplicaciones que interactúen con el clúster de DAX.

Cómo crear una Cuenta de AWS

Si no dispone de una Cuenta de AWS, siga estos pasos para crear una.

Procedimiento para registrarse en Cuenta de AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga las instrucciones que se le indiquen.

   Parte del procedimiento de registro consiste en recibir una llamada telefónica o mensaje de texto e indicar un código de verificación en el teclado del teléfono.

   Al registrarse en una Cuenta de AWS, se crea un Usuario raíz de la cuenta de AWS. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario y utilice únicamente el usuario raíz para realizar tareas que requieren acceso de usuario raíz.

AWS le enviará un correo electrónico de confirmación cuando complete el proceso de registro. Puede ver la actividad de la cuenta y administrarla en cualquier momento entrando en https://aws.amazon.com/ y seleccionando Mi cuenta.

Creación de un usuario con acceso administrativo

Después de registrarse para obtener una Cuenta de AWS, proteja su Usuario raíz de la cuenta de AWS, habilite AWS IAM Identity Center y cree un usuario administrativo para no usar el usuario raíz en las tareas cotidianas.

Protección de Usuario raíz de la cuenta de AWS

1. Inicie sesión en Consola de administración de AWS como propietario de la cuenta; para ello, elija Usuario raíz e introduzca el correo electrónico de su Cuenta de AWS. En la siguiente página, escriba su contraseña.

   Para obtener ayuda para iniciar sesión con el usuario raíz, consulte Iniciar sesión como usuario raíz en la Guía del usuario de AWS Sign-In.
2. Active la autenticación multifactor (MFA) para el usuario raíz.

   Para obtener instrucciones, consulte Habilitación de un dispositivo MFA virtual para su usuario raíz de la Cuenta de AWS (consola) en la Guía del usuario de IAM.

Creación de un usuario con acceso administrativo

1. Activar IAM Identity Center.

   Consulte las instrucciones en Activar AWS IAM Identity Center en la Guía del usuario de AWS IAM Identity Center.
2. En IAM Identity Center, conceda acceso administrativo a un usuario.

   Para ver un tutorial sobre cómo usar Directorio de IAM Identity Center como origen de identidad, consulte Configuración del acceso de los usuarios con el Directorio de IAM Identity Center predeterminado en la Guía del usuario de AWS IAM Identity Center.

Inicio de sesión como usuario con acceso de administrador

- Para iniciar sesión con el usuario de IAM Identity Center, use la URL de inicio de sesión que se envió a la dirección de correo electrónico cuando creó el usuario de IAM Identity Center.

Para obtener ayuda para iniciar sesión con un usuario de IAM Identity Center, consulte Inicio de sesión en el portal de acceso de AWS en la Guía del usuario de AWS Sign-In.

Concesión de acceso a usuarios adicionales

1. En IAM Identity Center, cree un conjunto de permisos que siga la práctica recomendada de aplicar permisos de privilegios mínimos.

   Para conocer las instrucciones, consulte Create a permission set en la Guía del usuario de AWS IAM Identity Center.
2. Asigne usuarios a un grupo y, a continuación, asigne el acceso de inicio de sesión único al grupo.

   Para conocer las instrucciones, consulte Add groups en la Guía del usuario de AWS IAM Identity Center.

Para dar acceso, agregue permisos a los usuarios, grupos o roles:

- Usuarios y grupos en AWS IAM Identity Center:

  Cree un conjunto de permisos. Siga las instrucciones de Creación de un conjunto de permisos en la Guía del usuario de AWS IAM Identity Center.
- Usuarios gestionados en IAM a través de un proveedor de identidades:

  Cree un rol para la federación de identidades. Siga las instrucciones descritas en Creación de un rol para un proveedor de identidad de terceros (federación) en la Guía del usuario de IAM.
- Usuarios de IAM:
  - Cree un rol que el usuario pueda aceptar. Siga las instrucciones descritas en Creación de un rol para un usuario de IAM en la Guía del usuario de IAM.
  - (No recomendado) Adjunte una política directamente a un usuario o agregue un usuario a un grupo de usuarios. Siga las instrucciones descritas en Adición de permisos a un usuario (consola) de la Guía del usuario de IAM.

Utilización del editor de política de JSON para la creación de una política

1. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en https://console.aws.amazon.com/iam/.
2. En el panel de navegación de la izquierda, elija Políticas.

   Si es la primera vez que elige Políticas, aparecerá la página Welcome to Managed Policies (Bienvenido a políticas administradas). Elija Comenzar.
3. En la parte superior de la página, seleccione Crear política.
4. En la sección Editor de políticas, seleccione la opción JSON.
5. Introduzca o pegue un documento de política de JSON. Para obtener más información sobre el lenguaje de la política de IAM, consulte Referencia de políticas JSON de IAM.
6. Resuelva las advertencias de seguridad, errores o advertencias generales que se generen durante la validación de la política y luego elija Siguiente.

   > **Note**
   >
   > Puede alternar entre las opciones Visual y JSON del editor en todo momento. No obstante, si realiza cambios o selecciona Siguiente en la opción Visual del editor, es posible que IAM reestructure la política, con el fin de optimizarla para el editor visual. Para obtener más información, consulte Reestructuración de política en la Guía del usuario de IAM.
7. (Opcional) Al crear o editar una política en la Consola de administración de AWS, se puede generar una plantilla de política JSON o YAML que se puede utilizar en plantillas de CloudFormation.

   Para ello, en el Editor de políticas, seleccione Acciones y, a continuación, Generar plantilla de CloudFormation. Para obtener más información sobre CloudFormation, consulte la Referencia de tipos de recursos de AWS Identity and Access Management en la Guía del usuario de AWS CloudFormation.
8. Cuando haya terminado de agregar permisos a la política, seleccione Siguiente.
9. En la página Revisar y crear, introduzca el Nombre de la política y la Descripción (opcional) para la política que está creando. Revise los Permisos definidos en esta política para ver los permisos que concede la política.

10. (Opcional) Agregar metadatos a la política al adjuntar las etiquetas como pares de clave-valor. Para obtener más información sobre el uso de etiquetas en IAM, consulte Etiquetas para recursos de AWS Identity and Access Management en la Guía del usuario de IAM.
11. Elija Crear política para guardar la nueva política.

Documento de política: copie y pegue el siguiente documento para crear la política de JSON.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        },
        {
            "Action": [
                "dynamodb:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Paso 3: configurar una instancia de Amazon EC2

Una vez que la instancia de Amazon EC2 esté disponible, puede iniciar sesión en la misma y prepararla para utilizarla.

> **Note**
>
> En los pasos siguientes se presupone que se va a conectar a la instancia de Amazon EC2 desde un equipo que ejecuta Linux. Para obtener información sobre otras maneras de conectarse, consulte Conexión con la instancia de Linux en la Guía del usuario de Amazon EC2.

Para configurar la instancia EC2

1. Abra la consola de Amazon EC2 en https://console.aws.amazon.com/ec2/.
2. Utilice el comando `ssh` para registrase en su instancia de Amazon EC2, como se muestra en el siguiente ejemplo.

   ```
   ssh -i my-keypair.pem ec2-user@public-dns-name
   ```

   Debe especificar el archivo de clave privada (archivo `.pem`) y el nombre de DNS público de la instancia. (Consulte Paso 1: lanzar una instancia de Amazon EC2.)

   El identificador de inicio de sesión es `ec2-user`. No se requiere contraseña.
3. Después de iniciar sesión en la instancia EC2, configure sus credenciales de AWS, tal y como se muestra a continuación. Ingrese el ID de clave de acceso y la clave secreta de AWS (de Paso 2: Crear un usuario y una política); a continuación, establezca el nombre de región predeterminada en su región actual. (En el siguiente ejemplo, el nombre de región predeterminada es). `us-west-2`.)

   ```
   aws configure

   AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
   AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   Default region name [None]: us-west-2
   Default output format [None]:
   ```

Después de lanzar y configurar su instancia de Amazon EC2, puede probar la funcionalidad de DAX utilizando una de las aplicaciones de ejemplo disponibles. Para obtener más información, consulte Paso 4: ejecutar un ejemplo de aplicación.

Paso 4: ejecutar un ejemplo de aplicación

Para ayudarle a probar la funcionalidad de Amazon DynamoDB Accelerator (DAX), puede ejecutar una de las aplicaciones de ejemplo disponibles en la instancia de Amazon EC2.

Temas

- Node.js y DAX
- SDK para Go de DAX
- Java y DAX
- .NET y DAX
- Python y DAX

Node.js y DAX

Configuración predeterminada de cliente para Node.js

Al configurar el cliente del SDK de JavaScript de DAX, puede personalizar varios parámetros para optimizar el rendimiento, la gestión de la conexión y la resiliencia a errores. En la siguiente tabla se describe la configuración predeterminada que controla la forma en que el cliente interactúa con el clúster de DAX, incluidos los valores de tiempo de espera, los mecanismos de reintento, la administración de credenciales y las opciones de supervisión del estado. Para obtener más información, consulte Operaciones de DynamoDBClient.

Valores predeterminados del cliente del SDK de JS de DAX

Parámetro	Tipo	Descripción
`region` opcional	`string`	La Región de AWS que se debe usar para el cliente de DAX (por ejemplo, “us-east-1”). Este parámetro es obligatorio si no se proporcio na a través de la variable de entorno.
`endpoint` obligatorio	`string`	El punto de conexión del clúster al que se conecta el SDK.

Parámetro	Tipo	Descripción
		Ejemplos: No cifrado: dax-cluster-name.region.amazonaws.com Cifrado: daxs://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
`requestTimeout` predeterminado: 6000 ms	`number`	Esto define el tiempo máximo que el cliente esperará una respuesta de DAX.
`writeRetries` predeterminado: 1	`number`	El número de reintentos que se realizarán para las solicitudes de escritura que fallan.
`readRetries` predeterminado: 1	`number`	El número de reintentos que se realizarán para las solicitudes de lectura que fallan.
`maxRetries` predeterminado: 1	`number`	El número máximo de reintentos para las solicitudes fallidas. Si se establecen readRetries/writeRetries, la configuración establecida en readRetries y writeRetries tiene prioridad sobre maxRetries.
`connectTimeout` predeterminado: 10000 ms	`number`	El tiempo de espera (en milisegundos) para establecer una conexión con cualquiera de los nodos del clúster.

Parámetro	Tipo	Descripción
`maxConcurrentConnections` predeterminado: 100	`number`	Limita el número total de conexiones simultáneas que una instancia de cliente puede crear por nodo en un clúster de DAX.
`maxRetryDelay` predeterminado: 7000 ms	`number`	Cuando el servidor DAX indica que es necesaria una recuperación estableciendo el indicador `waitForRecoveryBeforeRetrying` en verdadero, el cliente hará una pausa antes de reintentar los intentos. Durante estos periodos de recuperación, el parámetro `maxRetryDelay` determina el tiempo máximo de espera entre reintentos. Esta configuración específica de recuperación solo se aplica cuando el servidor DAX está en modo de recuperación. Para todos los demás escenarios, el comportamiento de reintentos sigue uno de dos patrones: o bien un retraso exponencial basado en el recuento de reintentos (regido por los parámetros `writeRetries`, `readRetries` o `maxRetries`), o bien un reintento inmediato dependiendo del tipo de excepción.

Parámetro	Tipo	Descripción
`credentials` opcional	`AwsCredentialIdentity` | `AwsCredentialIdentityProvider`	Las credenciales de AWS que se utilizarán para la autenticación de solicitudes. Esto puede proporcionarse como AwsCredentialIdentity o AwsCredentialIdentityProvider. Si no se proporciona, el AWS SDK utilizará automáticamente la cadena de proveedores de credenciales predeterminada. Ejemplo: `{ accessKeyId: 'AKIA...', secretAccessKey: '...', sessionToken: '...' }` * @default Usa la cadena del proveedor de credenciales de AWS predeterminada.
`healthCheckInterval` predeterminado: 5000 ms	`number`	El intervalo (en milisegundos) entre las comprobaciones de estado del clúster. Con un intervalo más bajo se realizarán las comprobaciones con mayor frecuencia.
`healthCheckTimeout` predeterminado: 1000 ms	`number`	El tiempo de espera (en milisegundos) para que se complete la comprobación de estado.

Parámetro	Tipo	Descripción
`skipHostnameVerification` predeterminado: false	`boolean`	Omitir la verificación del nombre de host de las conexiones TLS. Esto no tiene ningún impacto en los clústeres sin cifrar. De forma predeterminada, se realiza la verificación del nombre de host; si se establece en True, se omitirá la verificación. Asegúrese de comprender las implicaciones de desactivar esta opción, que es la imposibilidad de autenticar el clúster al que se está conectando.
`unhealthyConsecutiveErrorCount` predeterminado: 5	`number`	Establece el número de errores consecutivos necesarios para indicar que el nodo se encuentra en un estado incorrecto en el intervalo de comprobación de estado.
`clusterUpdateInterval` predeterminado: 4000 ms	`number`	Devuelve el intervalo entre sondeos de miembros del clúster para buscar cambios de pertenencia.
`clusterUpdateThreshold` predeterminado: 125	`number`	Devuelve el umbral por debajo del cual no se sondeará el clúster para buscar cambios de pertenencia.

Parámetro	Tipo	Descripción
`credentailProvider` opcional | predeterminado: nulo	AwsCredentialIdentityProvider	Proveedor definido por el usuario para las credencias les de AWS utilizadas para autenticar solicitudes a DAX.

Configuración de paginación para DaxDocument

Nombre	Tipo	Detalle
`client`	DaxDocument	Instancia del tipo de DaxDocument.
`pageSize`	número	Determina el número de elementos por página.
`startingToken` Opcional	any	LastEvaluatedKey de la respuesta anterior se puede utilizar para solicitudes posteriores.

Para obtener información sobre el uso de la paginación, consulte the section called “TryDax.js”.

Migración al SDK de Node.js de DAX V3

Esta guía de migración lo ayudará en la transición de las aplicaciones Node.js de DAX existentes. El nuevo SDK requiere Node.js 18 o superior e introduce varios cambios importantes en la forma en que estructurará el código de DynamoDB Accelerator. Esta guía lo guiará a través de las diferencias clave, incluidos los cambios de sintaxis, los nuevos métodos de importación y los patrones de programación asincrónica actualizados.

Uso de DAX de Node.js V2

```
const AmazonDaxClient = require('amazon-dax-client');
const AWS = require('aws-sdk');

var region = "us-west-2";
```

```
AWS.config.update({
  region: region,
});

var client = new AWS.DynamoDB.DocumentClient();

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  client = new AWS.DynamoDB.DocumentClient({ service: dax });
}

// Make Get Call using Dax
var params = {
    TableName: 'TryDaxTable',
    pk: 1,
    sk: 1
}
client.get(params, function (err, data) {
    if (err) {
        console.error(
            "Unable to read item. Error JSON:",
            JSON.stringify(err, null, 2)
          );
    } else {
        console.log(data);
    }
});
```

Uso de DAX de Node.js V3

Para utilizar Node.js V3 de DAX, la versión 18 o superior de Node es la versión preferida. Para cambiar a Node 18, utilice lo siguiente:

```
// Import AWS DAX V3
import { DaxDocument } from '@amazon-dax-sdk/lib-dax';

// Import AWS SDK V3 DynamoDBDocument ~ DocumentClient in V2
import { DynamoDBDocument } from '@aws-sdk/lib-dynamodb';
import { DynamoDBClient } from '@aws-sdk/client-dyanmodb';
```

```
// Create DynamoDBDocument
var client = DynamoDBDocument.from(new DynamoDB({region: 'us-west-2'});

// Override DynamoDBDocument Client with DaxDocument
if (process.argv.length > 2) {
  client = new DaxDocument({
    endpoints: [process.argv[2]],
    region: 'us-west-2',
  });
}

var params = {
    TableName: 'TryDaxTable',
    pk: 1,
    sk: 1
}
// Dax Shifted it's API Calls to await/promise
try {
  const results = await client.get(params);
  console.log(results);
} catch (err) {
  console.error(err)
}
```

El SDK de DAX para Node.js v3.x es compatible con el AWS SDK para Node.js v3.x. El SDK de DAX para Node.js v3.x admite el uso de clientes agregados. Tenga en cuenta que DAX no admite la creación de clientes básicos. Para obtener más información sobre las características no admitidas, consulte the section called “Características que no están en paridad con AWS SDK V3”.

Siga estos pasos para ejecutar la aplicación de ejemplo de Node.js en su instancia de Amazon EC2.

Para ejecutar el ejemplo de Node.js para DAX

1. Configure Node.js en su instancia ,de Amazon EC2 de la siguiente manera:

   a. Instale el administrador de la versión de nodo (nvm).

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash
   ```

   b. Use nvm para instalar Node.js.

   ```
   nvm install 18
   ```

c. Uso de nvm para utilizar Node 18

```
nvm use 18
```

d. Compruebe que Node.js está instalado y se ejecuta correctamente.

```
node -e "console.log('Running Node.js ' + process.version)"
```

Esto debería mostrar el mensaje siguiente.

```
Running Node.js v18.x.x
```

2. Instale el cliente Node.js de DaxDocument mediante el administrador del paquete de Node (npm).

```
npm install @amazon-dax-sdk/lib-dax
```

Código de muestra TryDax

Para evaluar las ventajas de rendimiento de DynamoDB Accelerator (DAX), siga estos pasos para ejecutar una prueba de muestra que compare los tiempos de operación de lectura entre DynamoDB estándar y un clúster de DAX.

1. Después de configurar el espacio de trabajo y agregar `lib-dax` como dependencia, copie the section called “TryDax.js” en el proyecto.
2. Ejecute el programa en el clúster de DAX. Para determinar el punto de enlace del clúster de DAX, elija una de las opciones siguientes:

   - En la consola de DynamoDB: elija su clúster de DAX. El punto de enlace del clúster se muestra en la consola, como en el siguiente ejemplo.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - En la AWS CLI: ingrese el siguiente comando.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     El punto de enlace del clúster se muestra en el resultado, como en el siguiente ejemplo.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

3. Ahora ejecute el programa; para ello, especifique el punto de conexión del clúster como parámetro de línea de comandos.

```
node TryDax.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Debería ver una salida similar a esta:

```
Attempting to create table; please wait...
Successfully created table. Table status: ACTIVE
Writing data to the table...
Writing 20 items for partition key:  1
Writing 20 items for partition key:  2
Writing 20 items for partition key:  3
...
Running GetItem Test
        Total time: 153555.10 µs - Avg time: 383.89 µs
        Total time: 44679.96 µs - Avg time: 111.70 µs
        Total time: 36885.86 µs - Avg time: 92.21 µs
        Total time: 32467.25 µs - Avg time: 81.17 µs
        Total time: 32202.60 µs - Avg time: 80.51 µs
Running Query Test
        Total time: 14869.25 µs - Avg time: 2973.85 µs
        Total time: 3036.31 µs - Avg time: 607.26 µs
        Total time: 2468.92 µs - Avg time: 493.78 µs
        Total time: 2062.53 µs - Avg time: 412.51 µs
        Total time: 2178.22 µs - Avg time: 435.64 µs
Running Scan Test
        Total time: 2395.88 µs - Avg time: 479.18 µs
        Total time: 2207.16 µs - Avg time: 441.43 µs
        Total time: 2443.14 µs - Avg time: 488.63 µs
        Total time: 2038.24 µs - Avg time: 407.65 µs
        Total time: 1972.17 µs - Avg time: 394.43 µs
Running Pagination Test
Scan Pagination
```

```
[
  { pk: 1, sk: 1, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 2, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 3, someData: 'XXXXXXXXXX' }
]
[
  { pk: 1, sk: 4, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 5, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 6, someData: 'XXXXXXXXXX' }
]
...
Query Pagination
[
  { pk: 1, sk: 1, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 2, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 3, someData: 'XXXXXXXXXX' }
]
[
  { pk: 1, sk: 4, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 5, someData: 'XXXXXXXXXX' },
  { pk: 1, sk: 6, someData: 'XXXXXXXXXX' }
]
...
Attempting to delete table; please wait...
Successfully deleted table.
```

Anote la información de tiempo. El número de microsegundos necesarios para las pruebas de `GetItem`, `Query` y `Scan`.

4. En este caso, ha ejecutado los programas en el clúster de DAX. Ahora, volverá a ejecutar el programa, esta vez en DynamoDB.

5. Ahora ejecute el programa de nuevo, pero esta vez, sin la URL del punto de conexión del clúster como parámetro de la línea de comandos.

```
node TryDax.js
```

Fíjese en el resultado y tome nota de la información sobre tiempos. Los tiempos transcurridos para las operaciones `GetItem`, `Query` y `Scan` deberían ser significativamente menores con DAX en comparación con DynamoDB.

Características que no están en paridad con AWS SDK V3

- Clientes básicos: Node.js de Dax V3 no admite clientes básicos.

```
const dynamoDBClient = new DynamoDBClient({ region: 'us-west-2' });
const regularParams = {
    TableName: 'TryDaxTable',
    Key: {
        pk: 1,
        sk: 1
    }
};
// The DynamoDB client supports the send operation.
const dynamoResult = await dynamoDBClient.send(new GetCommand(regularParams));

// However, the DaxDocument client does not support the send operation.
const daxClient = new DaxDocument({
    endpoints: ['your-dax-endpoint'],
    region: 'us-west-2',
});

const params = {
    TableName: 'TryDaxTable',
    Key: {
        pk: 1,
        sk: 1
    }
};

// This will throw an error - send operation is not supported for DAX. Please refer
 to documentation.
const result = await daxClient.send(new GetCommand(params));
console.log(result);
```

- Pila de middleware: Node.js de Dax V3 no admite el uso de funciones de middleware.

```
const dynamoDBClient = new DynamoDBClient({ region: 'us-west-2' });
// The DynamoDB client supports the middlewareStack.
dynamoDBClient.middlewareStack.add(
  (next, context) =>> async (args) => {
    console.log("Before operation:", args);
```

```
    const result = await next(args);
    console.log("After operation:", result);
    return result;
  },
  {
    step: "initialize", // or "build", "finalizeRequest", "deserialize"
    name: "loggingMiddleware",
  }
);

// However, the DaxDocument client does not support the middlewareStack.
const daxClient = new DaxDocument({
    endpoints: ['your-dax-endpoint'],
    region: 'us-west-2',
});

// This will throw an error - custom middleware and middlewareStacks are not
 supported for DAX. Please refer to documentation.
daxClient.middlewareStack.add(
  (next, context) => async (args) => {
    console.log("Before operation:", args);
    const result = await next(args);
    console.log("After operation:", result);
    return result;
  },
  {
    step: "initialize", // or "build", "finalizeRequest", "deserialize"
    name: "loggingMiddleware",
  }
);
```

TryDax.js

```
import { DynamoDB, waitUntilTableExists, waitUntilTableNotExists } from "@aws-sdk/
client-dynamodb";
import { DaxDocument, daxPaginateScan, daxPaginateQuery } from "@amazon-dax-sdk/lib-
dax";
import { DynamoDBDocument, paginateQuery, paginateScan } from "@aws-sdk/lib-dynamodb";

const region = "us-east-1";
const tableName = "TryDaxTable";
```

```
// Determine the client (DynamoDB or DAX)
let client = DynamoDBDocument.from(new DynamoDB({ region }));
if (process.argv.length > 2) {
  client = new DaxDocument({ region, endpoint: process.argv[2] });
}

// Function to create table
async function createTable() {
  const dynamodb = new DynamoDB({ region });
  const params = {
    TableName: tableName,
    KeySchema: [
      { AttributeName: "pk", KeyType: "HASH" },
      { AttributeName: "sk", KeyType: "RANGE" },
    ],
    AttributeDefinitions: [
      { AttributeName: "pk", AttributeType: "N" },
      { AttributeName: "sk", AttributeType: "N" },
    ],
    ProvisionedThroughput: { ReadCapacityUnits: 25, WriteCapacityUnits: 25 },
  };

  try {
    console.log("Attempting to create table; please wait...");
    await dynamodb.createTable(params);
    await waitUntilTableExists({ client: dynamodb, maxWaitTime: 30 }, { TableName:
 tableName });
    console.log("Successfully created table. Table status: ACTIVE");
  } catch (err) {
    console.error("Error in table creation:", err);
  }
}

// Function to insert data
async function writeData() {
  console.log("Writing data to the table...");
  const someData = "X".repeat(10);
  for (let ipk = 1; ipk <= 20; ipk++) {
    console.log("Writing 20 items for partition key: ", ipk)
    for (let isk = 1; isk <= 20; isk++) {
      try {
        await client.put({ TableName: tableName, Item: { pk: ipk, sk: isk,
 someData } });
```

```
      } catch (err) {
        console.error("Error inserting data:", err);
      }
    }
  }
}

// Function to test GetItem
async function getItemTest() {
  console.log("Running GetItem Test");
  for (let i = 0; i < 5; i++) {
    const startTime = performance.now();
    const promises = [];
    for (let ipk = 1; ipk <= 20; ipk++) {
      for (let isk = 1; isk <= 20; isk++) {
        promises.push(client.get({ TableName: tableName, Key: { pk: ipk, sk: isk } }));
      }
    }
    await Promise.all(promises);
    const endTime = performance.now();
    const iterTime = (endTime - startTime) * 1000;
    const totalTime = iterTime.toFixed(2).padStart(3, ' ');
    const avgTime = (iterTime / 400).toFixed(2).padStart(3, ' ');
    console.log(`\tTotal time: ${totalTime} \u00B5s - Avg time: ${avgTime} \u00B5s`);
  }
}

// Function to test Query
async function queryTest() {
  console.log("Running Query Test");
  for (let i = 0; i < 5; i++) {
    const startTime = performance.now();
    const promises = [];
    for (let pk = 1; pk <= 5; pk++) {
      const params = {
        TableName: tableName,
        KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
        ExpressionAttributeValues: { ":pkval": pk, ":skval1": 1, ":skval2": 2 },
      };
      promises.push(client.query(params));
    }
    await Promise.all(promises);
    const endTime = performance.now();
    const iterTime = (endTime - startTime) * 1000;
```

```
    const totalTime = iterTime.toFixed(2).padStart(3, ' ');
    const avgTime = (iterTime / 5).toFixed(2).padStart(3, ' ');
    console.log(`\tTotal time: ${totalTime} \u00B5s - Avg time: ${avgTime} \u00B5s`);
  }
}

// Function to test Scan
async function scanTest() {
  console.log("Running Scan Test");
  for (let i = 0; i < 5; i++) {
    const startTime = performance.now();
    const promises = [];
    for (let pk = 1; pk <= 5; pk++) {
      const params = {
        TableName: tableName,
        FilterExpression: "pk = :pkval and sk between :skval1 and :skval2",
        ExpressionAttributeValues: { ":pkval": pk, ":skval1": 1, ":skval2": 2 },
      };
      promises.push(client.scan(params));
    }
    await Promise.all(promises);
    const endTime = performance.now();
    const iterTime = (endTime - startTime) * 1000;
    const totalTime = iterTime.toFixed(2).padStart(3, ' ');
    const avgTime = (iterTime / 5).toFixed(2).padStart(3, ' ');
    console.log(`\tTotal time: ${totalTime} \u00B5s - Avg time: ${avgTime} \u00B5s`);
  }
}

// Function to test Pagination
async function paginationTest() {
  console.log("Running Pagination Test");
  console.log("Scan Pagination");
  const scanParams = { TableName: tableName };
  const paginator = process.argv.length > 2 ? daxPaginateScan : paginateScan;
  for await (const page of paginator({ client, pageSize: 3 }, scanParams)) {
    console.log(page.Items);
  }

  console.log("Query Pagination");
  const queryParams = {
    TableName: tableName,
    KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
    ExpressionAttributeValues: { ":pkval": 1, ":skval1": 1, ":skval2": 10 },
```

```
  };
  const queryPaginator = process.argv.length > 2 ? daxPaginateQuery : paginateQuery;
  for await (const page of queryPaginator({ client, pageSize: 3 }, queryParams)) {
    console.log(page.Items);
  }
}

// Function to delete the table
async function deleteTable() {
  const dynamodb = new DynamoDB({ region });
  console.log("Attempting to delete table; please wait...")
  try {
    await dynamodb.deleteTable({ TableName: tableName });
    await waitUntilTableNotExists({ client: dynamodb, maxWaitTime: 30 }, { TableName:
 tableName });
    console.log("Successfully deleted table.");
  } catch (err) {
    console.error("Error deleting table:", err);
  }
}

// Execute functions sequentially
(async function () {
  await createTable();
  await writeData();
  await getItemTest();
  await queryTest();
  await scanTest();
  await paginationTest();
  await deleteTable();
})();
```

SDK para Go de DAX

Siga este procedimiento para ejecutar la aplicación de ejemplo de SDK para Go de Amazon DynamoDB Accelerator (DAX) en su instancia de Amazon EC2.

Para ejecutar la muestra de SDK para Go para DAX

1. Configure SDK para Go en su instancia de Amazon EC2:

    a. Instale el lenguaje de programación Go (`Golang`).

```
sudo yum install -y golang
```

b. Compruebe que Golang está instalado y se ejecuta correctamente.

```
go version
```

Debería aparecer un mensaje como este.

```
go version go1.23.4 linux/amd64
```

2. Instale la aplicación Golang de ejemplo.

```
go get github.com/aws-samples/sample-aws-dax-go-v2
```

3. Ejecute los siguientes programas de Golang. El primer programa crea una tabla de DynamoDB denominada `TryDaxGoTable`. El segundo programa escribe datos en la tabla.

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command create-table
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command put-item
```

4. Ejecute los siguientes programas de Golang.

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command get-item
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command query
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command scan
```

Tome nota de la información de tiempo; es decir, del número de milisegundos necesarios para realizar las pruebas de `GetItem`, `Query` y `Scan`.

5. En el paso anterior, ha ejecutado los programas en el punto de enlace de DynamoDB. Ahora, ejecute los programas de nuevo, pero, esta vez, las operaciones `GetItem`, `Query` y `Scan` se procesan en el clúster de DAX.

   Para determinar el punto de enlace del clúster de DAX, elija una de las opciones siguientes:

   - En la consola de DynamoDB: elija su clúster de DAX. El punto de enlace del clúster se muestra en la consola, como en el siguiente ejemplo.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - En la AWS CLI: ingrese el siguiente comando.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     El punto de enlace del clúster se muestra en el resultado, como en el siguiente ejemplo.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     }
     ```

   Ahora, vuelva a ejecutar los programas, pero, esta vez, especifique el punto de enlace del clúster como parámetro en la línea de comandos.

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
    -service dax -command get-item -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
    -service dax -command query -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
    -service dax -command scan -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command paginated-scan -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command paginated-query -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go
 -service dax -command paginated-batch-get -endpoint my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com:8111
```

Fíjese en el resto del resultado y tome nota de la información sobre tiempos. Los tiempos transcurridos para las operaciones `GetItem`, `Query` y `Scan` deberían ser significativamente menores con DAX que con DynamoDB.

6. Ejecute el siguiente programa de Golang para eliminar `TryDaxGoTable`.

```
go run ~/go/pkg/mod/github.com/aws-samples/sample-aws-dax-go-v2@v1.0.0/try_dax.go -
service dynamodb -command delete-table
```

Características que no están en paridad con AWS SDK para Go SDK V2

Pila de middleware: Go de DAX V2 no admite el uso de pilas de middleware a través de opciones de la API. Para obtener más información, consulte Personalización de las solicitudes de cliente de AWS SDK para Go v2 con middleware.

Ejemplo:

```
// Custom middleware implementation
type customSerializeMiddleware struct{}
// ID returns the identifier for the middleware
func (m *customSerializeMiddleware) ID() string {
    return "CustomMiddleware"
}
// HandleSerialize implements the serialize middleware handler
func (m *customSerializeMiddleware) HandleSerialize(
    ctx context.Context,
    in middleware.SerializeInput,
```

```
    next middleware.SerializeHandler,
) (
    out middleware.SerializeOutput,
    metadata middleware.Metadata,
    err error,
) {
    // Add your custom logic here before the request is serialized
    fmt.Printf("Executing custom middleware for request: %v\n", in)
    // Call the next handler in the middleware chain
    return next.HandleSerialize(ctx, in)
}

func executeGetItem(ctx context.Context) error {
    client, err := initItemClient(ctx)
    if err != nil {
        os.Stderr.WriteString(fmt.Sprintf("failed to initialize client: %v\n", err))
        return err
    }

    st := time.Now()
    for c := 0; c < iterations; c++ {
        for i := 0; i < pkMax; i++ {
            for j := 0; j < skMax; j++ {
                // Create key using attributevalue.Marshal for type safety
                pk, err := attributevalue.Marshal(fmt.Sprintf("%s_%d", keyPrefix, i))
                if err != nil {
                    return fmt.Errorf("error marshaling pk: %v", err)
                }
                sk, err := attributevalue.Marshal(fmt.Sprintf("%d", j))
                if err != nil {
                    return fmt.Errorf("error marshaling sk: %v", err)
                }
                key := map[string]types.AttributeValue{
                    "pk": pk,
                    "sk": sk,
                }
                in := &dynamodb.GetItemInput{
                    TableName: aws.String(table),
                    Key:       key,
                }

                // Custom middleware option
                customMiddleware := func(o *dynamodb.Options) {
```

```
                    o.APIOptions = append(o.APIOptions, func(stack *middleware.Stack)
 error {
                        // Add custom middleware to the stack
                        return stack.Serialize.Add(&customSerializeMiddleware{},
 middleware.After)
                    })
                }

                // Apply options to the GetItem call
                out, err := client.GetItem(ctx, in, customMiddleware)
                if err != nil {
                    return err
                }
                writeVerbose(out)
            }
        }
    }
    d := time.Since(st)
    os.Stdout.WriteString(fmt.Sprintf("Total Time: %v, Avg Time: %v\n", d, d/
iterations))
    return nil
}
```

Salida:

```
failed to execute command: custom middleware through APIOptions is not supported in DAX
 client
exit status 1
```

Configuración predeterminada de cliente para Go

Esta guía lo guiará a través de las opciones de configuración que le permiten afinar el rendimiento del cliente de DAX, la administración de conexiones y el comportamiento de registro. Al comprender la configuración predeterminada y cómo personalizarla, puede optimizar la interacción de la aplicación Go con DAX.

En esta sección

- Valores predeterminados del cliente del SDK de Go de DAX
- Creación del cliente

Valores predeterminados del cliente del SDK de Go de DAX

Parámetro	Tipo	Descripción
`Region` obligatorio	`string`	La Región de AWS que se debe usar para el cliente de DAX (ejemplo: “us-east-1”). Este parámetro es obligatorio si no se proporciona a través del entorno.
`HostPorts` obligatorio	`[] string`	Lista de puntos de conexión del clúster de DAX a los que se conecta el SDK. Por ejemplo: Sin cifrar: dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com Cifrado: daxs://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
`MaxPendingConnectionsPerHost` predeterminado: 10	`number`	Número de intentos de conexión simultáneos. (Las conexiones pueden estar en proceso de establecerse simultáneamente).
`ClusterUpdateThreshold` predeterminado: 125 * time.Millisecond	`time.Duration`	El tiempo mínimo que debe transcurrir entre las actualizaciones del clúster.
`ClusterUpdateInterval`	`time.Duration`	El intervalo en el que el cliente actualizará automáticamente

Parámetro	Tipo	Descripción
predeterminado: 4 * time.Second		la información del clúster de DAX.
`IdleConnectionsReapDelay` predeterminado: 30 * time.Second	`time.Duration`	El intervalo en el que el cliente cerrará las conexiones inactivas en el cliente de DAX.
`ClientHealthCheckInterval` predeterminado: 5 * time.Second	`time.Duration`	El intervalo en el que el cliente realizará comprobaciones de estado en los puntos de conexión del clúster de DAX.
`Credentials` predeterminado	`aws.CredentialsProvider`	Las credenciales de AWS utilizadas por el cliente de DAX para autenticar las solicitudes al servicio de DAX. Consulte Credenciales y proveedores de credenciales.
`DialContext` predeterminado	`func`	Una función personalizada utilizada por el cliente de DAX para establecer conexiones con el clúster de DAX.

Parámetro	Tipo	Descripción
`SkipHostnameVerification` predeterminado: false	bool	Omitir la verificación del nombre de host de las conexiones TLS. Esta configuración solo afecta a los clústeres cifrados. Cuando se establece en True, desactiva la verificación del nombre de host. Deshabilitar la verificación significa que no puede autenticar la identidad del clúster al que se está conectando, lo que plantea riesgos de seguridad. De forma predeterminada, la verificación del nombre de host está habilitada.
`RouteManagerEnabled` predeterminado: false	`bool`	Este indicador se utiliza para eliminar las rutas que enfrentan errores de red.
`RequestTimeout` predeterminado: 60 * time.Second	`time.Duration`	Esto define el tiempo máximo que el cliente esperará una respuesta de DAX. Prioridad: tiempo de espera del contexto (si está establecido) > `RequestTimmeout` (si está establecido) > predeterminado 60 s `RequestTimeout`.
`WriteRetries` predeterminado: 2	`number`	El número de reintentos que se realizarán para las solicitudes de escritura que fallan.

Parámetro	Tipo	Descripción
`ReadRetries` predeterminado: 2	`number`	El número de reintentos que se realizarán para las solicitud es de lectura que fallan.
`RetryDelay` predeterminado: 0	`time.Duration`	El retraso para errores sin limitación (en segundos) de los reintentos cuando falla una solicitud.
`Logger` opcional	`logging.Logger`	El registrador es una interfaz para registrar entradas en determinadas clasificaciones.
`LogLevel` predeterminado: utils.LogOff	`number`	Este nivel de registro se define solo para DAX. Se puede importar con github.com/aws/aws-dax-go-v2/tree/main/dax/utils. `const ( LogOff LogLevelType = 0 LogDebug LogLevelType = 1 LogDebugWithReques tRetries LogLevelType = 2 )`

> **Note**
>
> Para `time.Duration`, la unidad predeterminada es el nanosegundo. Si no especificamos ninguna unidad para ningún parámetro, se considerará que es nanosegundos: `daxCfg.ClusterUpdateInterval = 10` significa 10 nanosegundos.

(`daxCfg.ClusterUpdateInterval = 10 * time.Millisecond` significa 10 milisegundos).

Creación del cliente

Para crear un cliente de DAX:

- Cree la configuración de DAX y, a continuación, cree el cliente de DAX mediante la configuración de DAX. Con esto, puede sobrescribir una configuración de DAX si es necesario.

```
import (
"github.com/aws/aws-dax-go-v2/dax/utils"
"github.com/aws/aws-dax-go-v2/dax"
)

// Non - Encrypted : 'dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com'.
// Encrypted : daxs://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com'.

config := dax.DefaultConfig()
config.HostPorts = []string{endpoint}
config.Region = region
config.LogLevel = utils.LogDebug
daxClient, err := dax.New(config)
```

Migración al SDK de Go de DAX V2

Esta guía de migración lo ayudará en la transición de las aplicaciones Go de DAX existentes.

Uso del SDK de Go de DAX V1

```
package main

import (
    "fmt"
    "os"

    "github.com/aws/aws-dax-go/dax"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/dynamodb"
)
```

```
func main() {
    region := "us-west-2"
    endpoint := "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"

    // Create session
    sess, err := session.NewSession(&aws.Config{
        Region: aws.String(region),
    })
    if err != nil {
        fmt.Printf("Failed to create session: %v\n", err)
        os.Exit(1)
    }

    // Configure DAX client
    cfg := dax.DefaultConfig()
    cfg.HostPorts = []string{endpoint}
    cfg.Region = region

    // Create DAX client
    daxClient, err := dax.New(cfg)
    if err != nil {
        fmt.Printf("Failed to create DAX client: %v\n", err)
        os.Exit(1)
    }
    defer daxClient.Close() // Don't forget to close the client

    // Create GetItem input
    input := &dynamodb.GetItemInput{
        TableName: aws.String("TryDaxTable"),
        Key: map[string]*dynamodb.AttributeValue{
            "pk": {
                N: aws.String("1"),
            },
            "sk": {
                N: aws.String("1"),
            },
        },
    }

    // Make the GetItem call
    result, err := daxClient.GetItem(input)
    if err != nil {
        fmt.Printf("Failed to get item: %v\n", err)
```

```
        os.Exit(1)
    }

    // Print the result
    fmt.Printf("GetItem succeeded: %+v\n", result)
}
```

Uso del SDK de Go de DAX V2

```
package main

import (
    "context"
    "fmt"
    "os"

    "github.com/aws/aws-dax-go-v2/dax"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
    "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
    "github.com/aws/aws-sdk-go-v2/aws"
)

func main() {
    ctx := context.Background()
    region := "us-west-2"
    endpoint := "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"

    // Create DAX config
    config := dax.DefaultConfig()
    // Specify Endpoint and Region
    config.HostPorts = []string{endpoint}
    config.Region = region
    // Enabling logging
    config.LogLevel = utils.LogDebug
    // Create DAX client
    daxClient, err := dax.New(config)
    if err != nil {
        fmt.Printf("Failed to create DAX client: %v\n", err)
        os.Exit(1)
    }
    defer daxClient.Close() // Don't forget to close the client
```

```
    // Create key using attributevalue.Marshal for type safety
    pk, err := attributevalue.Marshal(fmt.Sprintf("%s_%d", keyPrefix, i))
    if err != nil {
        return fmt.Errorf("error marshaling pk: %v", err)
    }
    sk, err := attributevalue.Marshal(fmt.Sprintf("%d", j))
    if err != nil {
        return fmt.Errorf("error marshaling sk: %v", err)
    }

    // Create GetItem input
    input := &dynamodb.GetItemInput{
        TableName: aws.String("TryDaxTable"),
        Key: map[string]types.AttributeValue{
             "pk": pk,
             "sk": sk,
        },
    }

    // Make the GetItem call
    result, err := daxClient.GetItem(ctx, input)
    if err != nil {
        fmt.Printf("Failed to get item: %v\n", err)
        os.Exit(1)
    }

    // Print the result
    fmt.Printf("GetItem succeeded: %+v\n", result)
}
```

Para obtener más detalles sobre el uso de la API, consulte las muestras de AWS.

Java y DAX

SDK para Java 2.x de DAX es compatible con SDK para Java 2.x de AWS. Está construido sobre Java 8+ e incluye el soporte para E/S no bloqueado. Para obtener información sobre el uso de DAX con AWS SDK para Java 1.x, consulte Uso de DAX con SDK de AWS para Java 1.x.

Uso del cliente como una dependencia de Maven

Siga estos pasos para utilizar el cliente para el SDK de DAX para Java en su aplicación como una dependencia.

1. Descargue e instale Apache Maven. Para obtener más información, consulte Downloading Apache Maven e Installing Apache Maven.
2. Agregue la dependencia de Maven del cliente al archivo POM (Project Object Model) de la aplicación. En este ejemplo, sustituya *x.x.x* por el número de versión real del cliente.

```
<!--Dependency:-->
<dependencies>
    <dependency>
        <groupId>software.amazon.dax</groupId>
        <artifactId>amazon-dax-client</artifactId>
        <version>x.x.x</version>
    </dependency>
</dependencies>
```

Código de muestra TryDax

Después de configurar el espacio de trabajo y agregar el SDK de DAX como dependencia, copie TryDax.java en el proyecto.

Ejecute el código utilizando este comando.

```
java -cp classpath TryDax
```

Debería ver un resultado similar a este.

```
Creating a DynamoDB client

Attempting to create table; please wait...
Successfully created table.  Table status: ACTIVE
Writing data to the table...
Writing 10 items for partition key: 1
Writing 10 items for partition key: 2
Writing 10 items for partition key: 3
...

Running GetItem and Query tests...
First iteration of each test will result in cache misses
Next iterations are cache hits

GetItem test - partition key 1-100 and sort keys 1-10
  Total time: 4390.240 ms - Avg time: 4.390 ms
```

```
  Total time: 3097.089 ms - Avg time: 3.097 ms
  Total time: 3273.463 ms - Avg time: 3.273 ms
  Total time: 3353.739 ms - Avg time: 3.354 ms
  Total time: 3533.314 ms - Avg time: 3.533 ms
Query test - partition key 1-100 and sort keys between 2 and 9
  Total time: 475.868 ms - Avg time: 4.759 ms
  Total time: 423.333 ms - Avg time: 4.233 ms
  Total time: 460.271 ms - Avg time: 4.603 ms
  Total time: 397.859 ms - Avg time: 3.979 ms
  Total time: 466.644 ms - Avg time: 4.666 ms

Attempting to delete table; please wait...
Successfully deleted table.
```

Tome nota de la información de tiempo; es decir, del número de milisegundos necesarios para realizar las pruebas de `GetItem` y `Query`. En este caso, ha ejecutado el programa en el punto de enlace de DynamoDB. Ahora volverá a ejecutar el programa, esta vez en el clúster de DAX.

Para determinar el punto de enlace del clúster de DAX, elija una de las siguientes opciones:

- En la consola de DynamoDB seleccione su clúster de DAX. El punto de enlace del clúster se muestra dentro de la consola, como en el siguiente ejemplo.

  ```
  dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
  ```

- Utilizando la AWS CLI, ingrese el siguiente comando.

  ```
  aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
  ```

  El puerto, la dirección y la URL del punto de enlace del clúster se muestran en el resultado, como en el siguiente ejemplo.

  ```
  {
      "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
      "Port": 8111,
      "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
  }
  ```

Ahora, vuelva a ejecutar el programa, pero, esta vez, especifique el la URL del punto de enlace del clúster como parámetro en la línea de comandos.

```
java -cp classpath TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Fíjese en el resultado y tome nota de la información sobre tiempos. Los tiempos transcurridos para las operaciones `GetItem` y `Query` deberían ser significativamente menores con DAX que con DynamoDB.

Métricas de SDK

Con SDK para Java 2.x de DAX, puede recopilar métricas sobre los clientes de servicio de su aplicación y analizar los resultados en Amazon CloudWatch. Para obtener más información, consulte Habilitar métricas de SDK.

> **Note**
>
> El SDK para Java de DAX solo recopila las métricas `ApiCallSuccessful` y `ApiCallDuration`.

TryDax.java

```
import java.util.Map;

import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.dax.ClusterDaxAsyncClient;
import software.amazon.dax.Configuration;

public class TryDax {
    public static void main(String[] args) throws Exception {
        DynamoDbAsyncClient ddbClient = DynamoDbAsyncClient.builder()
```

```
                .build();

        DynamoDbAsyncClient daxClient = null;
        if (args.length >= 1) {
            daxClient = ClusterDaxAsyncClient.builder()
                    .overrideConfiguration(Configuration.builder()
                            .url(args[0]) // e.g. dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com
                            .build())
                    .build();
        }

        String tableName = "TryDaxTable";

        System.out.println("Creating table...");
        createTable(tableName, ddbClient);

        System.out.println("Populating table...");
        writeData(tableName, ddbClient, 100, 10);

        DynamoDbAsyncClient testClient = null;
        if (daxClient != null) {
            testClient = daxClient;
        } else {
            testClient = ddbClient;
        }

        System.out.println("Running GetItem and Query tests...");
        System.out.println("First iteration of each test will result in cache misses");
        System.out.println("Next iterations are cache hits\n");

        // GetItem
        getItemTest(tableName, testClient, 100, 10, 5);

        // Query
        queryTest(tableName, testClient, 100, 2, 9, 5);

        System.out.println("Deleting table...");
        deleteTable(tableName, ddbClient);
    }

    private static void createTable(String tableName, DynamoDbAsyncClient client) {
        try {
            System.out.println("Attempting to create table; please wait...");
```

```
            client.createTable(CreateTableRequest.builder()
                    .tableName(tableName)
                    .keySchema(KeySchemaElement.builder()
                            .keyType(KeyType.HASH)
                            .attributeName("pk")
                            .build(), KeySchemaElement.builder()
                            .keyType(KeyType.RANGE)
                            .attributeName("sk")
                            .build())
                    .attributeDefinitions(AttributeDefinition.builder()
                            .attributeName("pk")
                            .attributeType(ScalarAttributeType.N)
                            .build(), AttributeDefinition.builder()
                            .attributeName("sk")
                            .attributeType(ScalarAttributeType.N)
                            .build())
                    .billingMode(BillingMode.PAY_PER_REQUEST)
                    .build()).get();
            client.waiter().waitUntilTableExists(DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            System.out.println("Successfully created table.");

        } catch (Exception e) {
            System.err.println("Unable to create table: ");
            e.printStackTrace();
        }
    }

    private static void deleteTable(String tableName, DynamoDbAsyncClient client) {
        try {
            System.out.println("\nAttempting to delete table; please wait...");
            client.deleteTable(DeleteTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            client.waiter().waitUntilTableNotExists(DescribeTableRequest.builder()
                    .tableName(tableName)
                    .build()).get();
            System.out.println("Successfully deleted table.");

        } catch (Exception e) {
            System.err.println("Unable to delete table: ");
            e.printStackTrace();
```

```
        }
    }

    private static void writeData(String tableName, DynamoDbAsyncClient client, int
pkmax, int skmax) {
        System.out.println("Writing data to the table...");

        int stringSize = 1000;
        StringBuilder sb = new StringBuilder(stringSize);
        for (int i = 0; i < stringSize; i++) {
            sb.append('X');
        }
        String someData = sb.toString();

        try {
            for (int ipk = 1; ipk <= pkmax; ipk++) {
                System.out.println(("Writing " + skmax + " items for partition key: " +
ipk));
                for (int isk = 1; isk <= skmax; isk++) {
                    client.putItem(PutItemRequest.builder()
                            .tableName(tableName)
                            .item(Map.of("pk", attr(ipk), "sk", attr(isk), "someData",
attr(someData)))
                            .build()).get();
                }
            }
        } catch (Exception e) {
            System.err.println("Unable to write item:");
            e.printStackTrace();
        }
    }

    private static AttributeValue attr(int n) {
        return AttributeValue.builder().n(String.valueOf(n)).build();
    }

    private static AttributeValue attr(String s) {
        return AttributeValue.builder().s(s).build();
    }

    private static void getItemTest(String tableName, DynamoDbAsyncClient client, int
pk, int sk, int iterations) {
        long startTime, endTime;
```

```
        System.out.println("GetItem test - partition key 1-" + pk + " and sort keys 1-"
+ sk);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            try {
                for (int ipk = 1; ipk <= pk; ipk++) {
                    for (int isk = 1; isk <= sk; isk++) {
                        client.getItem(GetItemRequest.builder()
                                .tableName(tableName)
                                .key(Map.of("pk", attr(ipk), "sk", attr(isk)))
                                .build()).get();
                    }
                }
            } catch (Exception e) {
                System.err.println("Unable to get item:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk * sk);
        }
    }

    private static void queryTest(String tableName, DynamoDbAsyncClient client, int pk,
int sk1, int sk2, int iterations) {
        long startTime, endTime;
        System.out.println("Query test - partition key 1-" + pk + " and sort keys
between " + sk1 + " and " + sk2);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            for (int ipk = 1; ipk <= pk; ipk++) {
                try {
                    // Pagination API for Query.
                    client.queryPaginator(QueryRequest.builder()
                            .tableName(tableName)
                            .keyConditionExpression("pk = :pkval and sk between :skval1
and :skval2")
                            .expressionAttributeValues(Map.of(":pkval", attr(ipk),
":skval1", attr(sk1), ":skval2", attr(sk2)))
                            .build()).items().subscribe((item) -> {
                    }).get();
                } catch (Exception e) {
                    System.err.println("Unable to query table:");
```

```
                    e.printStackTrace();
                }
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk);
        }
    }

    private static void printTime(long startTime, long endTime, int iterations) {
        System.out.format("\tTotal time: %.3f ms - ", (endTime - startTime) /
 (1000000.0));
        System.out.format("Avg time: %.3f ms\n", (endTime - startTime) / (iterations *
 1000000.0));
    }
}
```

.NET y DAX

Siga estos pasos para ejecutar el ejemplo de .NET en su instancia de Amazon EC2.

> **Note**
>
> En este tutorial se utiliza el SDK de .NET 9. Muestra cómo puede ejecutar un programa en su Amazon VPC predeterminada para acceder a su clúster de Amazon DynamoDB Accelerator (DAX). Funciona con el AWS SDK v4 para .NET. Para obtener detalles sobre los cambios en V4 e información sobre la migración, consulte Migración a la versión 4 del AWS SDK para .NET. Si lo prefiere, puede utilizar AWS Toolkit for Visual Studio para escribir una aplicación .NET e implementarla en su VPC.
> Para obtener más información, consulte Creación y desarrollo de las aplicaciones de Elastic Beanstalk en .NET utilizando Toolkit for Visual Studio de AWS en la Guía para desarrolladores de AWS Elastic Beanstalk.

Para ejecutar el ejemplo de .NET para DAX

1. Vaya a la página de descargas de Microsoft y descargue el SDK de .NET 9 para Linux más reciente. El archivo descargable es `dotnet-sdk-N.N.N-linux-x64.tar.gz`.
2. Extraiga los archivos del SDK.

   ```
   mkdir dotnet
   ```

```
tar zxvf dotnet-sdk-N.N.N-linux-x64.tar.gz -C dotnet
```

Sustituya *N.N.N* por el número de versión real del SDK de .NET (por ejemplo: `9.0.305`).

3. Verifica la instalación.

```
alias dotnet=$HOME/dotnet/dotnet
dotnet --version
```

Esto debería imprimir el número de versión del SDK de .NET.

> **Note**
>
> En lugar del número de versión, podría recibir el error siguiente:
> error: libunwind.so.8: cannot open shared object file: No such file or directory
> Para resolver el error, instale el paquete `libunwind`.
>
> ```
> sudo yum install -y libunwind
> ```
>
> Después de esto, debería poder ejecutar el comando `dotnet --version` sin errores.

4. Creación de un nuevo proyecto de .NET.

```
dotnet new console -o myApp
```

Esto requiere algunos minutos para llevar a cabo una configuración solo una vez. Cuando se complete, ejecute el proyecto de ejemplo.

```
dotnet run --project myApp
```

Debería recibir el siguiente mensaje: `Hello World!`

5. El archivo `myApp/myApp.csproj` contiene metadatos acerca de su proyecto. Para utilizar el cliente de DAX en su aplicación, modifique el archivo para que tenga un aspecto similar a lo siguiente.

```
<Project Sdk="Microsoft.NET.Sdk">
    <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>net9.0</TargetFramework>
```

```
    </PropertyGroup>
    <ItemGroup>
        <PackageReference Include="AWSSDK.DAX.Client" Version="*" />
    </ItemGroup>
</Project>
```

6. Descargue el código fuente del programa de ejemplo (archivo `.zip`).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

Cuando haya terminado la descarga, extraiga los archivos de código fuente.

```
unzip TryDax.zip
```

7. Ahora ejecute los programas de ejemplo de dotNet, uno cada vez. Para cada programa, copie su contenido en `myApp/Program.cs` y, a continuación, ejecute el proyecto `MyApp`.

Ejecute los siguientes programas de .NET. El primer programa crea una tabla de DynamoDB denominada `TryDaxTable`. El segundo programa escribe datos en la tabla.

```
cp TryDax/dotNet/01-CreateTable.cs myApp/Program.cs
dotnet run --project myApp

cp TryDax/dotNet/02-Write-Data.cs myApp/Program.cs
dotnet run --project myApp
```

8. A continuación, ejecute algunos programas para llevar a cabo operaciones `GetItem`, `Query` y `Scan` en su clúster de DAX. Para determinar el punto de enlace del clúster de DAX, elija una de las opciones siguientes:

- En la consola de DynamoDB: elija su clúster de DAX. El punto de enlace del clúster se muestra en la consola, como en el siguiente ejemplo.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- En la AWS CLI: ingrese el siguiente comando.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

El punto de enlace del clúster se muestra en el resultado, como en el siguiente ejemplo.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

Ahora, ejecute los programas siguientes, especificando el punto de enlace del clúster como parámetro en la línea de comandos. (Reemplace el punto de enlace del ejemplo por el punto de enlace de clúster de DAX real).

```
cp TryDax/dotNet/03-GetItem-Test.cs myApp/Program.cs
dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com

cp TryDax/dotNet/04-Query-Test.cs myApp/Program.cs
dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com

cp TryDax/dotNet/05-Scan-Test.cs myApp/Program.cs
dotnet run --project myApp dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
```

Tome nota de la información de tiempo; es decir, del número de milisegundos necesarios para realizar las pruebas de `GetItem`, `Query` y `Scan`.

9. Ejecute el siguiente programa de .NET para eliminar `TryDaxTable`.

```
cp TryDax/dotNet/06-DeleteTable.cs myApp/Program.cs
dotnet run --project myApp
```

Para obtener más información sobre estos programas, consulte las siguientes secciones:

- 01-CreateTable.cs
- 02-Write-Data.cs
- 03-GetItem-Test.cs
- 04-Query-Test.cs
- 05-Scan-Test.cs

- 06-DeleteTable.cs

01-CreateTable.cs

El programa `01-CreateTable.cs` crea una tabla (`TryDaxTable`). Los demás programas de .NET en esta sección dependerán de esta tabla.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            var request = new CreateTableRequest()
            {
                TableName = tableName,
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement{ AttributeName = "pk",KeyType = "HASH"},
                    new KeySchemaElement{ AttributeName = "sk",KeyType = "RANGE"}
                },
                AttributeDefinitions = new List<AttributeDefinition>() {
                    new AttributeDefinition{ AttributeName = "pk",AttributeType = "N"},
                    new AttributeDefinition{ AttributeName = "sk",AttributeType  = "N"}
                },
                ProvisionedThroughput = new ProvisionedThroughput()
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 10
                }
            };
```

```
            var response = await client.CreateTableAsync(request);

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

02-Write-Data.cs

El programa `02-Write-Data.cs` escribe datos de prueba en `TryDaxTable`.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            string someData = new string('X', 1000);
            var pkmax = 10;
            var skmax = 10;

            for (var ipk = 1; ipk <= pkmax; ipk++)
            {
                Console.WriteLine($"Writing {skmax} items for partition key: {ipk}");
                for (var isk = 1; isk <= skmax; isk++)
                {
                    var request = new PutItemRequest()
                    {
                        TableName = tableName,
                        Item = new Dictionary<string, AttributeValue>()
                       {
```

```
                            { "pk", new AttributeValue{N = ipk.ToString() } },
                            { "sk", new AttributeValue{N = isk.ToString() } },
                            { "someData", new AttributeValue{S = someData } }
                        }
                    };

                    var response = await client.PutItemAsync(request);
                }
            }

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

03-GetItem-Test.cs

El programa `03-GetItem-Test.cs` escribe realiza operaciones `GetItem` en `TryDaxTable`.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");

            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
            };
            var client = new ClusterDaxClient(clientConfig);
```

```
            var tableName = "TryDaxTable";

            var pk = 1;
            var sk = 10;
            var iterations = 5;

            var startTime = System.DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                for (var ipk = 1; ipk <= pk; ipk++)
                {
                    for (var isk = 1; isk <= sk; isk++)
                    {
                        var request = new GetItemRequest()
                        {
                            TableName = tableName,
                            Key = new Dictionary<string, AttributeValue>() {
                            {"pk", new AttributeValue {N = ipk.ToString()} },
                            {"sk", new AttributeValue {N = isk.ToString() } }
                        }
                        };
                        var response = await client.GetItemAsync(request);
                        Console.WriteLine($"GetItem succeeded for pk: {ipk},sk:
 {isk}");
                    }
                }
            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

04-Query-Test.cs

El programa `04-Query-Test.cs` escribe realiza operaciones `Query` en `TryDaxTable`.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");


            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
            };
            var client = new ClusterDaxClient(clientConfig);

            var tableName = "TryDaxTable";

            var pk = 5;
            var sk1 = 2;
            var sk2 = 9;
            var iterations = 5;

            var startTime = DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                var request = new QueryRequest()
                {
                    TableName = tableName,
                    KeyConditionExpression = "pk = :pkval and sk between :skval1
 and :skval2",
                    ExpressionAttributeValues = new Dictionary<string,
 AttributeValue>() {
                            {":pkval", new AttributeValue {N = pk.ToString()} },
```

```
                            {":skval1", new AttributeValue {N = sk1.ToString()} },
                            {":skval2", new AttributeValue {N = sk2.ToString()} }
                    }
                };
                var response = await client.QueryAsync(request);
                Console.WriteLine($"{i}: Query succeeded");

            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

05-Scan-Test.cs

El programa `05-Scan-Test.cs` escribe realiza operaciones `Scan` en `TryDaxTable`.

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.DAX;
using Amazon.DynamoDBv2.Model;

namespace ClientTest
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            string endpointUri = args[0];
            Console.WriteLine($"Using DAX client - endpointUri={endpointUri}");

            var clientConfig = new DaxClientConfig(endpointUri)
            {
                AwsCredentials = FallbackCredentialsFactory.GetCredentials()
```

```
            };
            var client = new ClusterDaxClient(clientConfig);

            var tableName = "TryDaxTable";

            var iterations = 5;

            var startTime = DateTime.Now;

            for (var i = 0; i < iterations; i++)
            {
                var request = new ScanRequest()
                {
                    TableName = tableName
                };
                var response = await client.ScanAsync(request);
                Console.WriteLine($"{i}: Scan succeeded");
            }

            var endTime = DateTime.Now;
            TimeSpan timeSpan = endTime - startTime;
            Console.WriteLine($"Total time: {timeSpan.TotalMilliseconds}
 milliseconds");

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

06-DeleteTable.cs

El programa `06-DeleteTable.cs` elimina `TryDaxTable`. Ejecute este programa después de haber terminado las pruebas.

```
using System;
using System.Threading.Tasks;
using Amazon.DynamoDBv2.Model;
using Amazon.DynamoDBv2;

namespace ClientTest
```

```
{
    class Program
    {
        public static async Task Main(string[] args)
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();

            var tableName = "TryDaxTable";

            var request = new DeleteTableRequest()
            {
                TableName = tableName
            };

            var response = await client.DeleteTableAsync(request);

            Console.WriteLine("Hit <enter> to continue...");
            Console.ReadLine();
        }
    }
}
```

Python y DAX

Siga este procedimiento para ejecutar la aplicación de ejemplo de Python en su instancia de Amazon EC2.

Para ejecutar la muestra de Python para DAX

1. Instale el cliente Python de DAX mediante la utilidad `pip`.

   ```
   pip install amazon-dax-client
   ```

2. Descargue el código fuente del programa de ejemplo (archivo `.zip`).

   ```
   wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   TryDax.zip
   ```

   Cuando haya terminado la descarga, extraiga los archivos de código fuente.

   ```
   unzip TryDax.zip
   ```

3. Ejecute los siguientes programas de Python. El primer programa crea una tabla de Amazon DynamoDB denominada `TryDaxTable`. El segundo programa escribe datos en la tabla.

```
python 01-create-table.py
python 02-write-data.py
```

4. Ejecute los siguientes programas de Python.

```
python 03-getitem-test.py
python 04-query-test.py
python 05-scan-test.py
```

   Tome nota de la información de tiempo; es decir, del número de milisegundos necesarios para realizar las pruebas de `GetItem`, `Query` y `Scan`.

5. En el paso anterior, ha ejecutado los programas en el punto de enlace de DynamoDB. Ahora, ejecute los programas de nuevo, pero, esta vez, las operaciones `GetItem`, `Query` y `Scan` se procesan en el clúster de DAX.

   Para determinar el punto de enlace del clúster de DAX, elija una de las opciones siguientes:

   - En la consola de DynamoDB: elija su clúster de DAX. El punto de enlace del clúster se muestra en la consola, como en el siguiente ejemplo.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

   - En la AWS CLI: ingrese el siguiente comando.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

     El punto de enlace del clúster se muestra en el resultado, como en este ejemplo.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

Vuelva a ejecutar los programas, pero, esta vez, especifique el punto de enlace del clúster como parámetro en la línea de comandos.

```
python 03-getitem-test.py dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
python 04-query-test.py dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
python 05-scan-test.py dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Fíjese en el resto del resultado y tome nota de la información sobre tiempos. Los tiempos transcurridos para las operaciones `GetItem`, `Query` y `Scan` deberían ser significativamente menores con DAX que con DynamoDB.

6. Ejecute el siguiente programa de Python para eliminar `TryDaxTable`.

```
python 06-delete-table.py
```

Para obtener más información sobre estos programas, consulte las siguientes secciones:

- 01-create-table.py
- 02-write-data.py
- 03-getitem-test.py
- 04-query-test.py
- 05-scan-test.py
- 06-delete-table.py

01-create-table.py

El programa `01-create-table.py` crea una tabla (`TryDaxTable`). Los demás programas de Python en esta sección dependerán de esta tabla.

```
import boto3


def create_dax_table(dyn_resource=None):
    """
    Creates a DynamoDB table.
```

```
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The newly created table.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table_name = "TryDaxTable"
    params = {
        "TableName": table_name,
        "KeySchema": [
            {"AttributeName": "partition_key", "KeyType": "HASH"},
            {"AttributeName": "sort_key", "KeyType": "RANGE"},
        ],
        "AttributeDefinitions": [
            {"AttributeName": "partition_key", "AttributeType": "N"},
            {"AttributeName": "sort_key", "AttributeType": "N"},
        ],
        "BillingMode": "PAY_PER_REQUEST",
    }
    table = dyn_resource.create_table(**params)
    print(f"Creating {table_name}...")
    table.wait_until_exists()
    return table


if __name__ == "__main__":
    dax_table = create_dax_table()
    print(f"Created table.")
```

02-write-data.py

El programa `02-write-data.py` escribe datos de prueba en `TryDaxTable`.

```
import boto3


def write_data_to_dax_table(key_count, item_size, dyn_resource=None):
    """
    Writes test data to the demonstration table.

    :param key_count: The number of partition and sort keys to use to populate the
                      table. The total number of items is key_count * key_count.
    :param item_size: The size of non-key data for each test item.
```

```
    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    some_data = "X" * item_size

    for partition_key in range(1, key_count + 1):
        for sort_key in range(1, key_count + 1):
            table.put_item(
                Item={
                    "partition_key": partition_key,
                    "sort_key": sort_key,
                    "some_data": some_data,
                }
            )
            print(f"Put item ({partition_key}, {sort_key}) succeeded.")


if __name__ == "__main__":
    write_key_count = 10
    write_item_size = 1000
    print(
        f"Writing {write_key_count*write_key_count} items to the table. "
        f"Each item is {write_item_size} characters."
    )
    write_data_to_dax_table(write_key_count, write_item_size)
```

03-getitem-test.py

El programa `03-getitem-test.py` escribe realiza operaciones `GetItem` en `TryDaxTable`. Este ejemplo se da para la región eu-west-1.

```
import argparse
import sys
import time
import amazondax
import boto3


def get_item_test(key_count, iterations, dyn_resource=None):
```

```
    """
    Gets items from the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param key_count: The number of items to get from the table in each iteration.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource('dynamodb')

    table = dyn_resource.Table('TryDaxTable')
    start = time.perf_counter()
    for _ in range(iterations):
        for partition_key in range(1, key_count + 1):
            for sort_key in range(1, key_count + 1):
                table.get_item(Key={
                    'partition_key': partition_key,
                    'sort_key': sort_key
                })
                print('.', end='')
                sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument(
        'endpoint_url', nargs='?',
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.")
    args = parser.parse_args()

    test_key_count = 10
    test_iterations = 50
    if args.endpoint_url:
        print(f"Getting each item from the table {test_iterations} times, "
              f"using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url,
 region_name='eu-west-1') as dax:
```

```
            test_start, test_end = get_item_test(
                test_key_count, test_iterations, dyn_resource=dax)
    else:
        print(f"Getting each item from the table {test_iterations} times, "
              f"using the Boto3 client.")
        test_start, test_end = get_item_test(
            test_key_count, test_iterations)
    print(f"Total time: {test_end - test_start:.4f} sec. Average time: "
          f"{(test_end - test_start)/ test_iterations}.")
```

04-query-test.py

El programa `04-query-test.py` escribe realiza operaciones `Query` en `TryDaxTable`.

```
import argparse
import time
import sys
import amazondax
import boto3
from boto3.dynamodb.conditions import Key


def query_test(partition_key, sort_keys, iterations, dyn_resource=None):
    """
    Queries the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param partition_key: The partition key value to use in the query. The query
                          returns items that have partition keys equal to this value.
    :param sort_keys: The range of sort key values for the query. The query returns
                      items that have sort key values between these two values.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    key_condition_expression = Key("partition_key").eq(partition_key) & Key(
        "sort_key"
    ).between(*sort_keys)
```

```
    start = time.perf_counter()
    for _ in range(iterations):
        table.query(KeyConditionExpression=key_condition_expression)
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.",
    )
    args = parser.parse_args()

    test_partition_key = 5
    test_sort_keys = (2, 9)
    test_iterations = 100
    if args.endpoint_url:
        print(f"Querying the table {test_iterations} times, using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url) as dax:
            test_start, test_end = query_test(
                test_partition_key, test_sort_keys, test_iterations, dyn_resource=dax
            )
    else:
        print(f"Querying the table {test_iterations} times, using the Boto3 client.")
        test_start, test_end = query_test(
            test_partition_key, test_sort_keys, test_iterations
        )

    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

05-scan-test.py

El programa `05-scan-test.py` escribe realiza operaciones `Scan` en `TryDaxTable`.

```
import argparse
import time
import sys
import amazondax
import boto3


def scan_test(iterations, dyn_resource=None):
    """
    Scans the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        table.scan()
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not used.",
    )
    args = parser.parse_args()
```

```
    test_iterations = 100
    if args.endpoint_url:
        print(f"Scanning the table {test_iterations} times, using the DAX client.")
        # Use a with statement so the DAX client closes the cluster after completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url) as dax:
            test_start, test_end = scan_test(test_iterations, dyn_resource=dax)
    else:
        print(f"Scanning the table {test_iterations} times, using the Boto3 client.")
        test_start, test_end = scan_test(test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

06-delete-table.py

El programa `06-delete-table.py` elimina `TryDaxTable`. Ejecute este programa después de haber terminado las pruebas de la funcionalidad de Amazon DynamoDB Accelerator (DAX).

```
import boto3


def delete_dax_table(dyn_resource=None):
    """
    Deletes the demonstration table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    table.delete()

    print(f"Deleting {table.name}...")
    table.wait_until_not_exists()


if __name__ == "__main__":
    delete_dax_table()
    print("Table deleted!")
```

Modificación de una aplicación existente para que use DAX

Si ya dispone de una aplicación de Java que utiliza Amazon DynamoDB, puede modificarla para que pueda acceder al clúster de DynamoDB Accelerator (DAX). No tiene que volver a escribir toda la aplicación porque el cliente Java de DAX es similar al cliente de bajo nivel de DynamoDB que se incluye en el SDK para Java 2.x de AWS. Para obtener más detalles vea Trabajar con elementos en DynamoDB.

> **Note**
>
> En este ejemplo se utiliza SDK para Java 2.x de AWS. Para la versión de SDK para Java 1.x heredada, consulte Modificación de una aplicación SDK para Java 1.x existente para que use DAX.

Para modificar el programa, reemplace el cliente de DynamoDB por un cliente de DAX.

```
Region region = Region.US_EAST_1;

// Create an asynchronous DynamoDB client
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
                .region(region)
                .build();

// Create an asynchronous DAX client
DynamoDbAsyncClient client = ClusterDaxAsyncClient.builder()
                .overrideConfiguration(Configuration.builder()
                    .url(<cluster url>) // for example, "dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com"
                    .region(region)
                    .addMetricPublisher(cloudWatchMetricsPub) // optionally enable SDK
 metric collection
                    .build())
                .build();
```

También puede utilizar la biblioteca de alto nivel que forma parte del SDK para Java 2.x de AWS, reemplazando el cliente DynamoDB por un cliente DAX.

```
Region region = Region.US_EAST_1;
DynamoDbAsyncClient dax = ClusterDaxAsyncClient.builder()
```

```
        .overrideConfiguration(Configuration.builder()
            .url(<cluster url>) // for example, "dax://my-cluster.l6fzcv.dax-
clusters.us-east-1.amazonaws.com"
            .region(region)
            .build())
        .build();

DynamoDbEnhancedAsyncClient enhancedClient = DynamoDbEnhancedAsyncClient.builder()
        .dynamoDbClient(dax)
        .build();
```

Para obtener más información, consulte Mapeo de elementos en tablas de DynamoDB.

Administración de los clústeres de DAX

En esta sección se abordan algunas de las tareas de administración comunes de los clústeres de Amazon DynamoDB Accelerator (DAX).

Temas

- Permisos de IAM para administrar un clúster de DAX
- Escalado de un clúster de DAX
- Personalización de las configuraciones de un clúster de DAX
- Configuración de los ajustes de TTL
- Compatibilidad con el etiquetado en DAX
- AWS CloudTrailIntegración de
- Eliminación de un clúster de DAX

Permisos de IAM para administrar un clúster de DAX

Al administrar clústeres de DAX mediante la Consola de administración de AWS en la AWS Command Line Interface (AWS CLI), recomendamos encarecidamente reducir el alcance de las acciones que los usuarios pueden llevar a cabo. De esta forma, contribuirá a mitigar los riesgos y a respetar el principio de seguridad de otorgar privilegios mínimos.

La siguiente explicación se centra en el control del acceso a las API de administración de DAX. Para obtener más información, consulte Amazon DynamoDB Accelerator en la Referencia de la API de Amazon DynamoDB.

> **Note**
>
> Para obtener información más detallada sobre la administración de permisos de AWS Identity and Access Management (IAM), consulte:
>
> - IAM y creación de clústeres de DAX: Creación de un clúster de DAX.
> - IAM y operaciones del plano de datos de DAX: Control de acceso a DAX.

Para las API de administración de DAX, no puede limitar las acciones de la API a un recurso concreto. El componente `Resource` debe establecerse en `"*"`. Esto difiere de las operaciones de la API del plano de datos de DAX, tales como `GetItem`, `Query` y `Scan`. Las operaciones del plano de datos se exponen a través del cliente de DAX y su alcance sí se puede limitar a recursos concretos.

A modo de ejemplo, tomemos el siguiente documento de política de IAM.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

Supongamos que el objetivo de esta política es permitir las llamadas a la API de administración de DAX para el clúster `DAXCluster01` exclusivamente.

Ahora, supongamos que un usuario emite el siguiente comando de la AWS CLI.

```
aws dax describe-clusters
```

Este comando producirá un error con la excepción Not Authorized, porque el alcance de la llamada a la API `DescribeClusters` subyacente no puede limitarse a un clúster específico. Aunque sintácticamente la política es válida, el comando no se ejecuta, porque el componente `Resource` debe establecerse en `"*"`. Sin embargo, si el usuario ejecuta un programa que envía llamadas del plano de datos de DAX (por ejemplo, `GetItem` o `Query`) a `DAXCluster01`, estas llamadas sí se llevarán a cabo correctamente. Esto se debe a que el alcance de las API de plano de datos de DAX se puede limitar a recursos concretos (en este caso, a)., `DAXCluster01`).

Si desea escribir una única política de IAM que abarque todos los supuestos (tanto las API de administración de DAX como las API de plano de datos de DAX), le sugerimos que incluya dos instrucciones diferenciadas en el documento de políticas. Una de las instrucciones deberá abordar los API de plano de datos de DAX y la otra, los API de administración de DAX.

En el ejemplo de política siguiente se muestra este enfoque. Observe que el alcance de la instrucción `DAXDataAPIs` se limita al recurso `DAXCluster01`, pero que el recurso correspondiente a `DAXManagementAPIs` ha de ser `"*"`. Las acciones que se muestran en cada instrucción son meramente ilustrativas. Puede personalizarlas como considere oportuno para su aplicación.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXDataAPIs",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]},
            {
            "Sid": "DAXManagementAPIs",
```

```
            "Action": [
                "dax:CreateParameterGroup",
                "dax:CreateSubnetGroup",
                "dax:DecreaseReplicationFactor",
                "dax:DeleteCluster",
                "dax:DeleteParameterGroup",
                "dax:DeleteSubnetGroup",
                "dax:DescribeClusters",
                "dax:DescribeDefaultParameters",
                "dax:DescribeEvents",
                "dax:DescribeParameterGroups",
                "dax:DescribeParameters",
                "dax:DescribeSubnetGroups",
                "dax:IncreaseReplicationFactor",
                "dax:ListTags",
                "dax:RebootNode",
                "dax:TagResource",
                "dax:UntagResource",
                "dax:UpdateCluster",
                "dax:UpdateParameterGroup",
                "dax:UpdateSubnetGroup"
                ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Escalado de un clúster de DAX

Existen dos opciones disponibles para escalar un clúster de DAX. La primera opción es el escalado horizontal, donde se agregan réplicas de lectura al clúster. La segunda opción es el escalado vertical, donde se seleccionan distintos tipos de nodos. Para obtener información sobre cómo elegir un tamaño de clúster y un tipo de nodo adecuados para la aplicación, consulte Guía de tamaño del clúster de DAX.

Escalado horizontal

Con el escalado horizontal, puede mejorar el rendimiento de las operaciones de lectura agregando más réplicas de lectura al clúster. Un único clúster de DAX admite hasta 10 réplicas de lectura, que puede agregar o eliminar mientras el clúster se está ejecutando.

Al agregar un nodo nuevo, debe sincronizar los datos de la caché de un nodo homólogo. Por lo tanto, el tiempo de adición varía en función del tamaño de la caché y de la carga de trabajo de la aplicación. Como práctica recomendada, le recomendamos que escale previamente su clúster para satisfacer los picos de tráfico previstos. Para obtener información sobre las directrices de dimensionamiento y las recomendaciones de seguimiento, consulte Guía de tamaño del clúster de DAX.

En los siguientes ejemplos de la AWS CLI se muestra cómo incrementar o reducir el número de nodos. El argumento `--new-replication-factor` especifica el número total de nodos del clúster. Uno de los nodos es el principal y los demás, las réplicas de lectura.

```
aws dax increase-replication-factor \
    --cluster-name MyNewCluster  \
    --new-replication-factor 5
```

```
aws dax decrease-replication-factor \
    --cluster-name MyNewCluster  \
    --new-replication-factor 3
```

> **Note**
>
> El estado del clúster cambia a `modifying` cuando se modifica el factor de replicación. El estado cambia a `available` cuando se complete la modificación.

Escalado vertical

Si tiene un gran conjunto de datos con los que trabajar, seguramente la aplicación se beneficiará si usa los tipos de nodos mayores. Los nodos mayores pueden permitir que el clúster almacene más datos en memoria, lo que reduce los errores de caché y mejorar el rendimiento de la aplicación en general. (Todos los nodos de un clúster de DAX deben ser del mismo tipo).

Si el clúster de DAX tiene una alta tasa de operaciones de escritura o errores de caché, la aplicación también podría beneficiarse de utilizar tipos de nodos mayores. Las operaciones de escritura y los errores de caché consumen recursos en el nodo principal del clúster. Por lo tanto, el uso de tipos de nodos mayores podría aumentar el rendimiento del nodo principal y, por lo tanto, permitir un mayor rendimiento para estos tipos de operaciones.

No puede modificar los tipos de los nodos en un clúster de DAX en ejecución. En lugar de ello, deberá crear un nuevo clúster con el tipo de nodo deseado. Para ver una lista de los tipos de nodos admitidos, consulte Nodos.

Para crear un nuevo clúster de DAX mediante la Consola de administración de AWS, CloudFormation, la AWS CLI o el SDK de AWS. (Para la AWS CLI, use el parámetro `--node-type` para especificar el tipo de nodo).

Personalización de las configuraciones de un clúster de DAX

Al crear un clúster de DAX, se utilizan las siguientes configuraciones predeterminados:

- Expulsión automática de la caché habilitada, con un período de vida (TTL) de 5 minutos
- Sin preferencias respecto a las zonas de disponibilidad
- Sin preferencias respecto a los tiempos de mantenimiento
- Notificaciones deshabilitadas

Para los nuevos clústeres, puede personalizar los ajustes en el momento de su creación. Para hacerlo en la Consola de administración de AWS, desactive Use default settings (Usar la configuración predeterminada) para modificar los siguientes ajustes:

- Network and Security (Red y seguridad): permite ejecutar nodos de clústeres de DAX individuales en distintas zonas de disponibilidad de la región de AWS actual. Si elige No Preference (Sin preferencias), los nodos se distribuirán automáticamente entre las zonas de disponibilidad.
- Parameter Group (Grupo de párametros): conjunto de parámetros con nombre que se aplican a todos los nodos del clúster. Puede utilizar un grupo de parámetros para especificar el comportamiento de la caché con respecto al TTL. Puede cambiar el valor de cualquier parámetro dado dentro de un grupo de parámetros (excepto el grupo de parámetros predeterminado `default.dax.1.0`) en cualquier momento.
- Maintenance Window (Ventana de mantenimiento): periodo semanal durante el cual se aplicarán los parches y las actualizaciones de software a los nodos del clúster. Puede elegir el día y la hora

de comienzo, así como la duración del tiempo de mantenimiento. Si elige No Preference (Sin preferencias), el periodo de mantenimiento se selecciona al azar dentro de un bloque de 8 horas por región. Para obtener más información, consulte Periodo de mantenimiento.

> **Note**
>
> Parameter Group (Grupo de parámetros) y Maintenance Window (Ventana de mantenimiento) también se pueden cambiar en cualquier momento en un clúster en ejecución.

Cuando se produce un evento de mantenimiento, DAX puede notificarle mediante Amazon Simple Notification Service (Amazon SNS). Para configurar las notificaciones, elija una opción en el selector Topic for SNS notification (Tema para notificación de SNS). Puede crear un tema de Amazon SNS nuevo o elegir uno disponible.

Para obtener información acerca de cómo configurar y suscribirse a un tema de Amazon SNS, consulte Introducción a Amazon SNS en la Guía del desarrollador de Amazon Simple Notification Service.

Configuración de los ajustes de TTL

DAX mantiene dos cachés para datos que lee de DynamoDB:

- Caché de elemento: para los elementos recuperados usando `GetItem` or `BatchGetItem`.
- Caché de consultas: para los conjuntos de resultados recuperados mediante `Query` o `Scan`.

Para obtener más información, consulte Caché de elementos y Caché de consultas.

El TTL predeterminado de cada una de estas cachés es de 5 minutos. Si desea utilizar otras configuraciones de TTL, puede lanzar un clúster de DAX con un grupo de parámetros personalizados. Para hacer esto en la consola, elija DAX | Parameter groups (DAX | Grupos de parámetros) en el panel de navegación.

También puede llevar a cabo estas tareas utilizando la AWS CLI. En el siguiente ejemplo se muestra cómo lanzar un nuevo clúster de DAX con un grupo de parámetros personalizados. En este ejemplo, el TTL de la caché de elementos se establece en 10 minutos y el de la caché de consultas, en 3 minutos.

1. Cree un nuevo grupo de parámetros.

```
aws dax create-parameter-group \
    --parameter-group-name custom-ttl
```

2. Establezca el TTL de la caché de elementos en 10 minutos (600 000 milisegundos).

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=record-ttl-millis,ParameterValue=600000"
```

3. Establezca el TTL de la caché de consultas en 3 minutos (180 000 milisegundos).

```
aws dax update-parameter-group \
    --parameter-group-name custom-ttl \
    --parameter-name-values "ParameterName=query-ttl-millis,ParameterValue=180000"
```

4. Compruebe que los parámetros se han configurado correctamente.

```
aws dax describe-parameters --parameter-group-name custom-ttl \
    --query "Parameters[*].[ParameterName,Description,ParameterValue]"
```

A partir de ahora, puede lanzar un nuevo clúster de DAX con este grupo de parámetros.

```
aws dax create-cluster \
    --cluster-name MyNewCluster \
    --node-type dax.r3.large \
    --replication-factor 3 \
    --iam-role-arn arn:aws:iam::123456789012:role/DAXServiceRole \
    --parameter-group custom-ttl
```

> **Note**
>
> No se puede modificar un grupo de parámetros que se encuentre en uso en una instancia de DAX en ejecución.

Compatibilidad con el etiquetado en DAX

Muchos servicios de AWS, incluido DynamoDB, admiten el etiquetado: la capacidad de etiquetar recursos con nombres definidos por el usuario. Puede asignar etiquetas a clústeres de DAX, lo que le permite identificar rápidamente todos los recursos de AWS que tienen la misma etiqueta o categorizar las facturas de AWS según las etiquetas que ha asignado.

Para obtener más información, consulte Cómo agregar etiquetas a los recursos en DynamoDB.

Mediante Consola de administración de AWS

Para administrar las etiquetas de los clústeres de DAX

1. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación, en DAX, elija Clusters (Clústeres).
3. Elija el clúster con el que desee trabajar.
4. Elija la pestaña Etiquetas. Aquí puede agregar, enumerar, editar o eliminar las etiquetas.

   Cuando la configuración sea la que desea, elija Apply Changes (Aplicar cambios).

Uso de AWS CLI

Cuando se utiliza la AWS CLI para administrar las etiquetas de los clústeres de DAX, es preciso determinar previamente el nombre de recurso de Amazon (ARN) del clúster. En el siguiente ejemplo se muestra cómo determinar el ARN de un clúster denominado `MyDAXCluster`.

```
aws dax describe-clusters \
    --cluster-name MyDAXCluster \
    --query "Clusters[*].ClusterArn"
```

En el resultado, el ARN tendrá un aspecto similar a este: `arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster`

En el siguiente ejemplo se muestra cómo etiquetar el clúster.

```
aws dax tag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tags="Key=ClusterUsage,Value=prod"
```

Para enumerar todas las etiquetas de un clúster.

```
aws dax list-tags \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster
```

Para eliminar una etiqueta, hay que especificar su clave.

```
aws dax untag-resource  \
    --resource-name arn:aws:dax:us-west-2:123456789012:cache/MyDAXCluster \
    --tag-keys ClusterUsage
```

AWS CloudTrailIntegración de

DAX se integra con AWS CloudTrail, lo que permite auditar las actividades de los clústeres de DAX. Puede utilizar registros de CloudTrail para ver todos los cambios realizados en el nivel de clúster. También puede los cambios en los componentes de los clústeres, como los nodos, los grupos de subredes y los grupos de parámetros. Para obtener más información, consulte Registrar las operaciones de DynamoDB mediante AWS CloudTrail.

Eliminación de un clúster de DAX

Si ya no usa un clúster de DAX, debe eliminarlo para que no se le cobre por los recursos no utilizados.

Puede eliminar un clúster de DAX mediante la consola o la AWS CLI. A continuación se muestra un ejemplo.

```
aws dax delete-cluster --cluster-name mydaxcluster
```

Supervisión de DynamoDB Accelerator

El monitoreo es una parte importante del mantenimiento de la fiabilidad, la disponibilidad y el rendimiento de Amazon DynamoDB Accelerator (DAX) y sus soluciones de AWS. Debe recopilar datos de monitoreo de todas las partes de su solución de AWS para que le resulte más sencillo depurar cualquier error que se produzca en distintas partes del código, en caso de que ocurra.

Antes de comenzar a monitorear DAX, debe crear un plan de monitoreo que incluya respuestas a las siguientes preguntas:

- ¿Cuáles son los objetivos de la supervisión?
- ¿Qué recursos va a supervisar?
- ¿Con qué frecuencia va a supervisar estos recursos?
- ¿Qué herramientas de supervisión va a utilizar?
- ¿Quién se encargará de realizar las tareas de supervisión?
- ¿Quién debería recibir una notificación cuando surjan problemas?

Temas

- Herramientas de supervisión para Acelerador de DynamoDB
- Supervisión con Amazon CloudWatch
- Registro de operaciones de DAX con AWS CloudTrail

Herramientas de supervisión para Acelerador de DynamoDB

AWS proporciona herramientas que puede utilizar para monitorear Amazon DynamoDB Accelerator (DAX). Puede configurar algunas de estas herramientas para que realicen la labor de monitorización automáticamente; sin embargo, otras requieren intervención manual. Le recomendamos que automatice las tareas de supervisión en la medida de lo posible.

Temas

- Herramientas de monitoreo automatizadas
- Herramientas de monitoreo manuales

Herramientas de monitoreo automatizadas

Puede utilizar las siguientes herramientas de monitoreo automatizadas para vigilar DAX e informar cuando haya algún problema:

- Alarmas de Amazon CloudWatch: vigile una métrica durante un periodo de tiempo especificado y realice una o varias acciones según el valor que tenga la métrica en comparación con un determinado umbral durante una serie de periodos de tiempo. La acción es una notificación enviada a un tema de Amazon Simple Notification Service (Amazon SNS) o a una política de Amazon EC2 Auto Scaling. Las alarmas de CloudWatch no invocan acciones tan solo por tener un estado determinado; es necesario que el estado haya cambiado y se mantenga durante un

número específico de periodos. Para obtener más información, consulte Supervisión de métricas en DynamoDB con Amazon CloudWatch.

- Amazon CloudWatch Logs: monitoree, almacene y obtenga acceso a los archivos de registro de AWS CloudTrail u otras fuentes. Para obtener más información, consulte Monitoring Log Files en la Guía del usuario de Amazon CloudWatch.
- Eventos de Amazon CloudWatch: seleccione los eventos y diríjalos hacia uno o varios flujos o funciones de destino para realizar cambios, capturar información de estado y aplicar medidas correctivas. Para obtener más información, consulte ¿Qué son los Eventos de Amazon CloudWatch? en la guía del usuario de Amazon CloudWatch.
- Monitoreo de registros de AWS CloudTrail: comparta archivos de registro entre cuentas, monitoree los archivos de registro de CloudTrail en tiempo real enviándolos a CloudWatch Logs, escriba aplicaciones de procesamiento de registros en Java y compruebe que los archivos de registro no hayan cambiado después de que CloudTrail los entregara. Para obtener más información, consulte Uso de archivos de registro de CloudTrail en la Guía del usuario de AWS CloudTrail.

Herramientas de monitoreo manuales

Otra parte importante del monitoreo de DAX implica el monitoreo manual de los elementos que no cubren las alarmas de CloudWatch. DAX, CloudWatch, Trusted Advisor y otros paneles de Consola de administración de AWS proporcionan una vista rápida del estado de su entorno de AWS. Le recomendamos que también verifique los archivos de registro en DAX.

- El panel de DAX muestra lo siguiente:
  - Estado de los servicios
- La página principal de CloudWatch muestra lo siguiente:
  - Alarmas y estado actual
  - Gráficos de alarmas y recursos
  - Estado de los servicios

  Además, puede utilizar CloudWatch para hacer lo siguiente:
  - Crear paneles personalizados para supervisar los servicios que le importan.
  - Realizar un gráfico con los datos de las métricas para resolver problemas y descubrir tendencias.
  - Buscar y examinar todas sus métricas de recursos de AWS.
  - Crear y editar las alarmas de notificación de problemas.

Supervisión con Amazon CloudWatch

Puede monitorear DynamoDB Accelerator (DAX) mediante Amazon CloudWatch, que recopila y procesa los datos sin procesar de DAX para convertirlos en métricas legibles y casi en tiempo real. Estas estadísticas se registran durante un periodo de dos semanas. Puede obtener acceso a información histórica y disponer de una mejor perspectiva sobre el rendimiento de su aplicación web o servicio. De forma predeterminada, los datos de las métricas de DAX se envían a CloudWatch automáticamente. Para obtener más información, consulte ¿Qué es Amazon CloudWatch? en la guía del usuario de Amazon Cloudwatch.

Temas

- ¿Cómo utilizo las métricas de DAX?
- Visualización de dimensiones y métricas de DAX
- Crear alarmas de CloudWatch para monitorear a DAX
- Monitoreo de la producción

¿Cómo utilizo las métricas de DAX?

Las métricas reportadas por DAX proporcionan información que puede analizar de diferentes maneras. En la siguiente lista se indican algunos usos frecuentes de las métricas. Se trata de sugerencias que puede usar como punto de partida y no de una lista completa.

¿Cómo...?	Métricas relevantes
Determinar si se ha producido algún error del sistema	Monitorice `FaultRequestCount` para determinar si alguna solicitud ha dado lugar a un código HTTP 500 (error del servidor). Esto puede indicar un error interno del servicio de DAX o un error HTTP 500 en la métrica SystemErrors de la tabla subyacente.
Determinar si se ha producido algún error del usuario	Monitorice `ErrorRequestCount` para determinar si alguna solicitud ha dado lugar a un código HTTP 400 (error del cliente). Si ve que el número de errores aumenta, tal vez desee investigar y asegurarse de que envía solicitudes del cliente correctas.
Determinar si se ha producido algún error de la caché	Monitorice `ItemCacheMisses` para determinar el número de veces que no se encontró un elemento en la caché, y

¿Cómo...?	Métricas relevantes
	`QueryCacheMisses` y `ScanCacheMisses` para determinar el número de veces que no se encontró el resultado de una consulta o examen en la caché.
Monitorizar la tasa de aciertos de caché	Utilice CloudWatch Metric Math para definir una métrica de tasa de aciertos de la caché con expresiones matemáticas. Por ejemplo, para la caché de elementos, puede utilizar la expresión m1/SUM([m1, m2])*100, donde m1 es la métrica `ItemCacheHits` y m2 es la métrica `ItemCacheMisses` del clúster. Para las cachés de consulta y análisis, puede utilizar el mismo patrón con la métrica de caché de consultas y análisis correspondiente.

Visualización de dimensiones y métricas de DAX

Cuando usted interactúa con Amazon DynamoDB, este envía las siguientes métricas y dimensiones a Amazon CloudWatch. Puede utilizar los siguientes procedimientos para cer las métricas de DynamoDB Accelerator (DAX).

Para ver las métricas (consola)

Las métricas se agrupan en primer lugar por el espacio de nombres de servicio y, a continuación, por las diversas combinaciones de dimensiones dentro de cada espacio de nombres.

1. Abra la consola de CloudWatch en https://console.aws.amazon.com/cloudwatch/.
2. En el panel de navegación, seleccione Métricas.
3. Seleccione el espacio de nombres de DAX.

Para ver las métricas de (AWS CLI)

- En el símbolo del sistema, ejecute el siguiente comando.

```
aws cloudwatch list-metrics --namespace "AWS/DAX"
```

Dimensiones y métricas de DAX

En las siguientes secciones se detallan las métricas y dimensiones que DAX envía a CloudWatch.

Métricas de DAX

Las siguientes métricas están disponibles en DAX. DAX solo envía métricas a CloudWatch cuando tienen un valor distinto de cero.

> **Note**
>
> CloudWatch agrega las siguientes métricas de DAX a intervalos de un minuto:
>
> - `CPUUtilization`
> - `CacheMemoryUtilization`
> - `NetworkBytesIn`
> - `NetworkBytesOut`
> - `BaselineNetworkBytesInUtilization`
> - `BaselineNetworkBytesOutUtilization`
> - `NetworkPacketsIn`
> - `NetworkPacketsOut`
> - `GetItemRequestCount`
> - `BatchGetItemRequestCount`
> - `BatchWriteItemRequestCount`
> - `DeleteItemRequestCount`
> - `PutItemRequestCount`
> - `UpdateItemRequestCount`
> - `TransactWriteItemsCount`
> - `TransactGetItemsCount`
> - `ItemCacheHits`
> - `ItemCacheMisses`
> - `QueryCacheHits`
> - `QueryCacheMisses`
> - `ScanCacheHits`

- ScanCacheMisses
- TotalRequestCount
- ErrorRequestCount
- FaultRequestCount
- FailedRequestCount
- QueryRequestCount
- ScanRequestCount
- ClientConnections
- EstimatedDbSize
- EvictedSize
- CPUCreditUsage
- CPUCreditBalance
- CPUSurplusCreditBalance
- CPUSurplusCreditsCharged

No todas las estadísticas, tales como `Average` o `Sum`, son aplicables a todas las métricas. Sin embargo, todos estos valores están disponibles a través de la consola de DAX o mediante la consola de CloudWatch, la AWS CLI, o los SDK de AWS para todas las métricas. En la siguiente tabla, cada métrica tiene una lista de estadísticas válidas y aplicables.

Métrica	Descripción
`CPUUtilization`	El porcentaje de utilización de CPU del nodo o del clúster. Unidades: `Percent` Estadísticas válidas: • `Minimum` • `Maximum` • `Average`
`CacheMemoryUtilization`	Porcentaje de memoria caché disponible que está utilizando la caché de elemento y la caché de consulta en el nodo o clúster.

Métrica	Descripción
	Los datos almacenados en caché comienzan a ser expulsados antes de que la utilización de la memoria llegue al 100% (vea la métrica `EvictedSize`). Si `CacheMemoryUtilization` alcanza el 100% en cualquier nodo, las solicitudes de escritura serán sometidas a una limitación controlada y debe considerar cambiar a un clúster con un tipo de nodo más grande. Unidades: `Percent` Estadísticas válidas: • `Minimum` • `Maximum` • `Average`
`NetworkBytesIn`	El número de bytes recibidos en todas las interfaces de red por el nodo o el clúster. Unidades: `Bytes` Estadísticas válidas: • `Minimum` • `Maximum` • `Average`

Métrica	Descripción
NetworkBytesOut	El número de bytes enviados en todas las interfaces de red por el nodo o el clúster. Esta métrica identifica el volumen de tráfico de red saliente en términos del número de bytes en un solo nodo o clúster. Unidades: Bytes Estadísticas válidas: • Minimum • Maximum • Average
BaselineNetworkBytesInUtilization	El porcentaje del ancho de banda de la red de referencia consumido en un momento dado para el tráfico de entrada. Como referencia, el 50 % significa que se está utilizando la mitad del ancho de banda de la red disponible para el tráfico de entrada. Unidades: Percent Estadísticas válidas: • Minimum • Maximum • Average • SampleCount • Sum

Métrica	Descripción
`BaselineNetworkBytesOutUtilization`	El porcentaje del ancho de banda de la red de referencia consumido en un momento determinado para el tráfico de salida. Como referencia, el 50 % significa que se está utilizando la mitad del ancho de banda de la red disponible para el tráfico de salida. Unidades: `Percent` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`NetworkPacketsIn`	El número de paquetes recibidos en todas las interfaces de red por el nodo o el clúster. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average`

Métrica	Descripción
`NetworkPacketsOut`	El número de paquetes enviados en todas las interfaces de red por el nodo o el clúster. Esta métrica identifica el volumen de tráfico saliente en términos del número de paquetes en un solo nodo o clúster. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average`
`GetItemRequestCount`	El número de solicitudes `GetItem` manejadas por el nodo o el clúster. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descripción
`BatchGetItemRequestCount`	El número de solicitudes `BatchGetItem` manejadas por el nodo o el clúster. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`BatchWriteItemRequestCount`	El número de solicitudes `BatchWriteItem` manejadas por el nodo o el clúster. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descripción
`DeleteItemRequestCount`	El número de solicitudes `DeleteItem` manejadas por el nodo o el clúster. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`PutItemRequestCount`	El número de solicitudes `PutItem` manejadas por el nodo o el clúster. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descripción
`UpdateItemRequestCount`	El número de solicitudes `UpdateItem` manejadas por el nodo o el clúster. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`TransactWriteItemsCount`	El número de solicitudes `TransactWriteItems` manejadas por el nodo o el clúster. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descripción
`TransactGetItemsCount`	El número de solicitudes `TransactGetItems` manejadas por el nodo o el clúster. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ItemCacheHits`	Cantidad de veces que el nodo o el clúster devolvieron un elemento desde la caché. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descripción
`ItemCacheMisses`	Cantidad de veces que un elemento no estaba en la caché del nodo o clúster y tuvo que recuperarse de DynamoDB. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`QueryCacheHits`	Cantidad de veces que se devolvió el resultado de una consulta desde la caché del nodo o clúster. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descripción
QueryCacheMisses	Cantidad de veces que el resultado de una consulta no estaba en la caché de nodo o clúster y tuvo que recuperarse de DynamoDB. Unidades: Count Estadísticas válidas: • Minimum • Maximum • Average • SampleCount • Sum
ScanCacheHits	Cantidad de veces que se devolvió el resultado de un análisis desde la caché del nodo o clúster. Unidades: Count Estadísticas válidas: • Minimum • Maximum • Average • SampleCount • Sum

Métrica	Descripción
`ScanCacheMisses`	Cantidad de veces que el resultado de un análisis no estaba en la caché del nodo o clúster y tuvo que recuperarse de DynamoDB. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`TotalRequestCount`	Cantidad total de solicitudes manejadas por el nodo o el clúster. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descripción
`ErrorRequestCount`	Cantidad total de solicitudes que dieron lugar a un error de usuario notificado por el nodo o el clúster. Se incluyen las solicitudes que fueron sometidas a una limitación controlada por el nodo o el clúster. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`ThrottledRequestCount`	Cantidad total de solicitudes sometidas a una limitación controlada por el nodo o el clúster. Las solicitudes que fueron sometidas a una limitación cotrolada por DynamoDB no se incluyen y se pueden monitorear mediante las métricas de DynamoDB. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descripción
`FaultRequestCount`	Cantidad total de solicitudes que dieron lugar a un error interno notificado por el nodo o el clúster. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`FailedRequestCount`	Cantidad total de solicitudes que dieron lugar a un error notificado por el nodo o el clúster. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descripción
QueryRequestCount	Cantidad de solicitudes de consulta manejadas por el nodo o el clúster. Unidades: Count Estadísticas válidas: • Minimum • Maximum • Average • SampleCount • Sum
ScanRequestCount	Cantidad de solicitudes de análisis manejadas por el nodo o el clúster. Unidades: Count Estadísticas válidas: • Minimum • Maximum • Average • SampleCount • Sum

Métrica	Descripción
`ClientConnections`	Cantidad de conexiones simultáneas realizadas por los clientes al nodo o clúster. Unidades: `Count` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`
`EstimatedDbSize`	Una aproximación de la cantidad de datos almacenados en la caché de elemento y la caché de consulta por el nodo o el clúster. Unidades: `Bytes` Estadísticas válidas: • `Minimum` • `Maximum` • `Average`

Métrica	Descripción
`EvictedSize`	La cantidad de datos expulsados por el nodo o el clúster para dejar espacio para los datos solicitados recientemente. Si la tasa de faltas aumenta y ve que esta métrica también crece, probablemente significa que su conjunto de trabajo ha aumentado. Debe considerar cambiar a un clúster con un tipo de nodo más grande. Unidades: `Bytes` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `Sum`

Métrica	Descripción
`CPUCreditUsage`	La cantidad de créditos de CPU gastados por el nodo para la utilización de la CPU. Un crédito de CPU equivale a una vCPU ejecutándose al 100% de utilización durante un minuto o una combinación equivalente de unidades de vCPU, utilización y tiempo (por ejemplo, una vCPU ejecutándose al 50% durante dos minutos o dos vCPU ejecutándose al 25% durante dos minutos). Las métricas de créditos de CPU solo están disponibles cada cinco minutos. Si especifica un periodo superior a cinco minutos, use la estadística `Sum` en lugar de la `Average`. Unidades: `Credits (vCPU-minutes)` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descripción
`CPUCreditBalance`	La cantidad de créditos de la CPU obtenidos que un nodo ha acumulado desde que se lanzó o se inició. Los créditos se acumulan en el saldo de créditos una vez obtenidos y se eliminan del saldo de créditos cuando se gastan. El saldo de créditos tiene un límite máximo, determinado por el tamaño del nodo de DAX. Una vez que se ha alcanzado el límite, los nuevos créditos obtenidos se descartarán. Los créditos de `CPUCreditBalance` están disponibles para que el nodo los gaste para ampliar la utilización de la CPU por encima de la línea de base. Unidades: `Credits (vCPU-minutes)` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

Métrica	Descripción
CPUSurplusCreditBalance	La cantidad de créditos sobrantes que ha gastado un nodo de DAX cuando su valor CPUCreditBalance es igual a cero. El valor de CPUSurplusCreditBalance se compensa con los créditos de CPU obtenidos. Si la cantidad de créditos sobrantes supera el número máximo de créditos que el nodo puede ganar en un periodo de 24 horas, los créditos sobrantes gastados por encima del máximo implican un cargo adicional. Unidades: Credits (vCPU-minutes) Estadísticas válidas: • Minimum • Maximum • Average • SampleCount • Sum

Métrica	Descripción
`CPUSurplusCreditsCharged`	La cantidad de créditos sobrantes gastados que no se han compensado con créditos de CPU obtenido y, por lo tanto, implican un cargo adicional. Se cobran créditos sobrantes gastados cuando los créditos sobrantes gastados superan el número máximo de créditos que el nodo puede obtener en un periodo de 24 horas. Los créditos sobrantes gastados por encima de la cantidad máxima se cobran al final de la hora o cuando se elimina el nodo. Unidades: `Credits (vCPU-minutes)` Estadísticas válidas: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `Sum`

> **Note**
>
> Las métricas `CPUCreditUsage`, `CPUCreditBalance`, `CPUSurplusCreditBalance` y `CPUSurplusCreditsCharged` solo están disponibles para los nodos T3.

Dimensiones de métricas de DAX

Las métricas de DAX se califican según los valores de cuenta, ID de clúster o la combinación de ID de clúster e ID de nodo. Puede usar la consola de CloudWatch junto con cualquier dimensión de la tabla siguiente para recuperar datos de DAX.

Dimensión	Descripción
`Account`	Proporciona estadísticas agregadas en todos los nodos de una cuenta.
`ClusterId`	Limita los datos a un clúster.
`ClusterId, NodeId`	Limita los datos a un nodo dentro de un clúster.

Crear alarmas de CloudWatch para monitorear a DAX

Puede crear una alarma de Amazon CloudWatch que envíe un mensaje de Amazon Simple Notification Service (Amazon SNS) cuando la alarma cambia de estado. Una alarma vigila una métrica determinada durante el periodo especificado. Realiza una o varias acciones según el valor de la métrica con respecto a un umbral dado durante varios períodos de tiempo. La acción es una notificación que se envía a un tema de Amazon SNS o a una política de Amazon EC2 Auto Scaling. Las alarmas invocan acciones únicamente para los cambios de estado prolongados. Las alarmas de CloudWatch no invocan acciones simplemente porque se encuentren en un estado determinado. El estado debe haber cambiado y debe mantenerse durante el número de periodos especificado.

¿Cómo puedo recibir notificaciones sobre los errores de la caché de consultas?

1. Crear un tema de Amazon SNS, `arn:aws:sns:us-west-2:522194210714:QueryMissAlarm`.

Para obtener más información, consulte Configurar Amazon Simple Notification Service en la Guía del usuario de Amazon CloudWatch.

2. Cree la alarma.

```
aws cloudwatch put-metric-alarm \
    --alarm-name QueryCacheMissesAlarm  \
    --alarm-description "Alarm over query cache misses" \
    --namespace AWS/DAX \
    --metric-name QueryCacheMisses  \
    --dimensions Name=ClusterID,Value=myCluster \
    --statistic Sum \
    --threshold 8 \
    --comparison-operator GreaterThanOrEqualToThreshold \
    --period 60 \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:us-west-2:522194210714:QueryMissAlarm
```

3. Pruebe la alarma.

```
aws cloudwatch set-alarm-state --alarm-name QueryCacheMissesAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name QueryCacheMissesAlarm --state-reason
 "initializing" --state-value ALARM
```

> **Note**
>
> Puede aumentar o reducir el umbral de manera que tenga sentido para su organización. También puede usar CloudWatch Metric Math para definir una métrica de porcentaje de errores de la caché y establecer una alarma que se active con esa métrica.

¿Cómo puedo recibir notificaciones si las solicitudes producen un error interno en el clúster?

1. Crear un tema de Amazon SNS, `arn:aws:sns:us-west-2:123456789012:notify-on-system-errors`.

Para obtener más información, consulte Configurar Amazon Simple Notification Service en la Guía del usuario de Amazon CloudWatch.

2. Cree la alarma.

```
aws cloudwatch put-metric-alarm \
    --alarm-name FaultRequestCountAlarm \
    --alarm-description "Alarm when a request causes an internal error" \
    --namespace AWS/DAX \
    --metric-name FaultRequestCount \
    --dimensions Name=ClusterID,Value=myCluster \
    --statistic Sum \
    --threshold 0 \
    --comparison-operator GreaterThanThreshold \
    --period 60 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:notify-on-system-errors
```

3. Pruebe la alarma.

```
aws cloudwatch set-alarm-state --alarm-name FaultRequestCountAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name FaultRequestCountAlarm --state-reason
 "initializing" --state-value ALARM
```

Monitoreo de la producción

Debe establecer un punto de referencia del rendimiento normal de DAX en su entorno. Para ello se mide el rendimiento en distintos momentos y bajo distintas condiciones de carga. Cuando monitoree DAX, debe tener en cuenta el almacenamiento de los datos históricos de monitoreo. Estos datos almacenados le ofrecen un punto de referencia con el que comparar los datos de rendimiento actuales, identificar los patrones de rendimiento normales y las anomalías de rendimiento, así como desarrollar métodos de resolución de problemas.

Para establecer una línea de referencia, como mínimo, debe monitorear los siguientes elementos durante las pruebas de carga y en producción.

- La utilización de CPU y las solicitudes de limitación controlada, de modo que pueda determinar si podría necesitar utilizar un tipo de nodo mayor en el clúster. La utilización de CPU del clúster está disponible a través de la métrica `CPUUtilization` de CloudWatch. La estadística media de esta métrica proporciona una vista del uso medio de la CPU en todos los nodos del clúster. Para tomar decisiones sobre el escalado del clúster, le recomendamos que utilice la estadística máxima, que es la utilización máxima en todos los nodos.

  > **Note**
  >
  > AWS ha mejorado el grado de detalle de la métrica `CPUUtilization`. Es posible que se observen cambios en la métrica entre el 17 de mayo de 2024 y el 22 de junio de 2024.

- La latencia de operación (medida en el lado del cliente) debe mantenerse de forma coherente dentro de los requisitos de latencia de la aplicación.
- Las tasas de error deben permanecer bajas, como se ve en la métricas de CloudWatch `ErrorRequestCount`, `FaultRequestCount` y `FailedRequestCount`.
- Consumo de bytes de red, para que pueda determinar si debe usar más nodos o un tipo de nodo más grande en el clúster. Para supervisar el consumo, puede configurar alertas en métricas `BaselineNetworkBytesInUtilization` y `BaselineNetworkBytesOutUtilization` disponibles en CloudWatch, que indiquen el porcentaje de consumo del ancho de banda de la red disponible para el tipo de instancia, para el tráfico de entrada y salida, respectivamente.
- El uso de la memoria caché y el tamaño desalojado, para que pueda determinar si el tipo de nodo del clúster tiene memoria suficiente para contener el conjunto de trabajo y si no, cambiar a un tipo de nodo más grande.

  > **Note**
  >
  > En caso de que se produzca un gran número de errores y escrituras en la memoria caché, el uso de la memoria caché puede aumentar hasta un 100 % y provocar un tiempo de inactividad de la disponibilidad.

- Conexiones cliente, para que pueda monitorizar cualquier pico insólito en las conexiones con el clúster.

Registro de operaciones de DAX con AWS CloudTrail

Amazon DynamoDB Accelerator (DAX) se integra con AWS CloudTrail, un servicio que proporciona un registro de las acciones llevadas a cabo por un usuario, un rol o un servicio de AWS en DAX.

Para obtener más información sobre DAX y CloudTrail, consulte la sección de DynamoDB Accelerator (DAX) en Registrar las operaciones de DynamoDB mediante AWS CloudTrail.

Instancias ampliables DAX T3/T2

DAX le permite elegir entre instancias de rendimiento fijo (como R4, R5 y R7) e instancias de rendimiento ampliable (como T2 y T3). Las instancias de rendimiento ampliable proporcionan un nivel base de rendimiento de la CPU con la posibilidad de ampliarse por encima del nivel básico cuando sea necesario.

El rendimiento básico y la capacidad de ampliarse sobre él se rigen por los créditos de CPU. Las instancias de rendimiento ampliables acumulan créditos de CPU de forma continua, a una velocidad determinada por el tamaño de la instancia, cuando la carga de trabajo está por debajo del umbral de línea base. Estos créditos se pueden consumir cuando la carga de trabajo aumenta. Un crédito de CPU proporciona el desempeño de un núcleo de CPU completo durante un minuto.

Muchas cargas de trabajo no necesitan niveles altos de CPU de manera constante, pero se benefician significativamente al tener acceso completo a CPU muy rápidas cuando las necesitan. Las instancias de rendimiento ampliable están diseñadas específicamente para estos casos de uso. Si su base de datos necesita un alto rendimiento de CPU constante, le recomendamos que utilice instancias de rendimiento fijo.

Familia de instancias DAX T2

Las instancias DAX T2 son instancias de rendimiento ampliable de uso general que proporcionan un nivel base de rendimiento de CPU con la posibilidad de ampliarse por encima del nivel básico. Las instancias T2 son una buena opción para cargas de trabajo de pruebas y desarrollo que necesitan previsibilidad de precios. Las instancias DAX T2 están configuradas para el modo estándar, lo que significa que si la instancia se está quedando sin créditos acumulados, la utilización de CPU se reduce gradualmente hasta el nivel de referencia. Para obtener más información sobre el modo estándar, consulte Modo estándar para las instancias de rendimiento ampliable en la Guía del usuario de Amazon EC2.

Familia de instancias DAX T3

Las instancias DAX T3 son la próxima generación de instancias ampliables de uso general que proporcionan un nivel básico de rendimiento de CPU con la posibilidad de ampliar el uso de CPU en cualquier momento durante el tiempo que sea necesario. Las instancias T3 ofrecen un equilibrio entre recursos informáticos, de memoria y de red, y son ideales para cargas de trabajo con un uso moderado de CPU que experimentan picos temporales en su uso. Las instancias DAX T3 están configuradas para el modo ilimitado, lo que significa que pueden ampliarse más allá de la línea base en una ventana de 24 horas con cargo adicional. Para obtener más información sobre el modo ilimitado, consulte Modo ilimitado para las instancias de rendimiento ampliable en la guía del usuario de Amazon EC2.

Las instancias DAX T3 pueden mantener un alto rendimiento de CPU durante el tiempo que lo requiera una carga de trabajo. Para mayoría de las cargas de trabajo de uso general, las instancias T3 proporcionan un amplio rendimiento sin cargos adicionales. El precio por hora de la instancia T3 cubre automáticamente todos los picos de uso provisionales cuando la utilización media de CPU de una instancia T3 está a la par o menos que la base de referencia en una ventana de 24 horas.

Por ejemplo, una instancia `dax.t3.small` recibe créditos de forma continua a una velocidad de 24 créditos de CPU por hora. Esta capacidad proporciona un rendimiento básico equivalente al 20% de un núcleo de CPU (20% × 60 minutos = 12 minutos). Si la instancia no utiliza los créditos que recibe, se almacenan en su saldo de crédito de CPU hasta un máximo de 576 créditos de CPU. Cuando la instancia `t3.small` necesita ampliarse a más del 20% de un núcleo, extrae de su saldo de crédito de CPU para manejar este aumento automáticamente.

Mientras que las instancias DAX T2 están restringidas al rendimiento de la línea base una vez que el saldo de crédito de la CPU se reduce a cero, las instancias de DAX T3 pueden ampliar la línea base incluso cuando su saldo de crédito de CPU es cero. Para la gran mayoría de las cargas de trabajo, donde la utilización media de CPU está a la par o por debajo del rendimiento base, el precio por hora básico para `t3.small` cubre todas las ampliaciones de CPU. Si la instancia se ejecuta a un promedio de utilización de CPU del 25% (5% por encima de la línea base) durante un período de 24 horas después de que su saldo de crédito de CPU se haya extraído a cero, se le cobrará 11,52 centavos USD adicionales (9,6 centavos USD/vCPU por hora × 1 vCPU × 5% × 24 horas). Consulte Precio de Amazon DynamoDB para obtener más información sobre los precios.

Control de acceso a DAX

DynamoDB Accelerator (DAX) está diseñado para funcionar conjuntamente con DynamoDB, con el fin de agregar de forma transparente una capa de almacenamiento en caché a las aplicaciones. Sin embargo, DAX y DynamoDB tienen mecanismos distintos de control de acceso. Aunque ambos servicios utilizan AWS Identity and Access Management (IAM) para implementar sus respectivas políticas de seguridad, los modelos de seguridad de DAX y DynamoDB son distintos.

Recomendamos encarecidamente que conozca ambos modelos de seguridad para poder implementar las medidas de seguridad adecuadas en las aplicaciones que utilizan DAX.

En esta sección se describen los mecanismos de control de acceso proporcionados por DAX y se facilitan ejemplos de políticas de IAM que puede adaptar a sus necesidades.

Con DynamoDB, puede crear políticas de IAM que limiten las acciones que el usuario puede llevar a cabo con los recursos de DynamoDB individuales. Por ejemplo, puede crear un rol de usuario que únicamente permita al usuario realizar acciones de solo lectura en una tabla de DynamoDB determinada. (Para obtener más información, consulte). Identity and Access Management en Amazon DynamoDB.) En comparación, el modelo de seguridad de DAX se centra en la seguridad del clúster y en la capacidad de éste para llevar a cabo las acciones de la API de DynamoDB en su nombre.

> **Warning**
>
> Si está utilizando los roles y las políticas de IAM para restringir el acceso a los datos de las tablas de DynamoDB, el uso de DAX puede alterar esas políticas. Por ejemplo, un usuario podría tener acceso a una tabla de DynamoDB mediante DAX aunque no tuviese acceso explícito a ella si el acceso se llevase a cabo directamente a través de DynamoDB. Para obtener más información, consulte Identity and Access Management en Amazon DynamoDB. DAX no aplica la separación de nivel de usuario de los datos en DynamoDB. En lugar de ello, los usuarios heredan los permisos de la política de IAM del clúster de DAX cuando obtienen acceso a ese clúster. Por lo tanto, al obtener acceso a las tablas de DynamoDB a través de DAX, los únicos controles de acceso que surten efecto son los permisos de la política de IAM del clúster de DAX. No se reconoce ningún otro permiso.
>
> Si requiere aislamiento, recomendamos crear clústeres de DAX adicionales y que defina en consecuencia el alcance la política IAM para cada clúster. Por ejemplo, podría crear varios clústeres de DAX y permitir que cada uno de ellos obtenga acceso a una sola tabla.

Rol de servicio de IAM para DAX

Al crear un clúster de DAX, es preciso asociarlo con un rol de IAM. Esto es lo que se denomina rol de servicio del clúster.

Supongamos que desea crear un nuevo clúster de DAX denominado DAXCluster01. Podría crear un rol de servicio denominado DAXServiceRole y asociarlo con DAXCluster01. La política para DAXServiceRole definiría las acciones de DynamoDB que DAXCluster01 podría llevar a cabo en nombre de los usuarios que interaccionen con DAXCluster01.

Cuando se crea un rol de servicio, es preciso especificar una relación de confianza entre DAXServiceRole y el servicio DAX propiamente dicho. Una relación de confianza determina qué entidades pueden asumir un rol y utilizar sus permisos. A continuación se muestra un ejemplo de documento de relación de confianza para DAXServiceRole:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "dax.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Esta relación de confianza permite que un clúster de DAX asuma DAXServiceRole y lleve a cabo llamadas al API de DynamoDB en su nombre.

Las acciones de la API de DynamoDB permitidas se describen en un documento de la política de IAM que se adjunta a DAXServiceRole. A continuación se muestra el ejemplo de documento de política.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DaxAccessPolicy",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:PutItem",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
            ]
        }
    ]
}
```

Esta política permite a DAX realizar todas las acciones de la API de DynamoDB necesarias en una tabla de DynamoDB. La acción `dynamodb:DescribeTable` es necesaria para que DAX mantenga metadatos sobre la tabla, y las demás son acciones de lectura y escritura realizadas en los elementos de la tabla. La tabla, denominada `Books`, está en la región us-west-2 y es propiedad del ID de cuenta de AWS `123456789012`.

> **Note**
>
> DAX apoya los mecanismos para evitar el problema del suplente confuso durante el acceso entre servicios. Para obtener más información, consulte El problema del suplente confuso en la Guía del usuario de IAM.

Política de IAM que permite el acceso a un clúster de DAX

Después de crear un clúster de DAX, es preciso conceder permisos a un usuario, para que este pueda obtener acceso al clúster de DAX.

Por ejemplo, supongamos que desea conceder acceso a DAXCluster01 a una usuaria cuyo nombre es Alice. En primer lugar, crearía la política de IAM (AliceAccessPolicy) que define los clústeres de DAX y las acciones del API de DAX a los que el destinatario puede obtener acceso. Posteriormente, le concedería acceso adjuntando esta política a la usuaria Alice.

El siguiente documento de política concede acceso pleno al destinatario en DAXCluster01.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "dax:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
            ]
        }
    ]
}
```

El documento de política permite acceder al clúster de DAX, pero no concede ningún permiso de DynamoDB. (Los permisos de DynamoDB los concede la función del servicio de DAX).

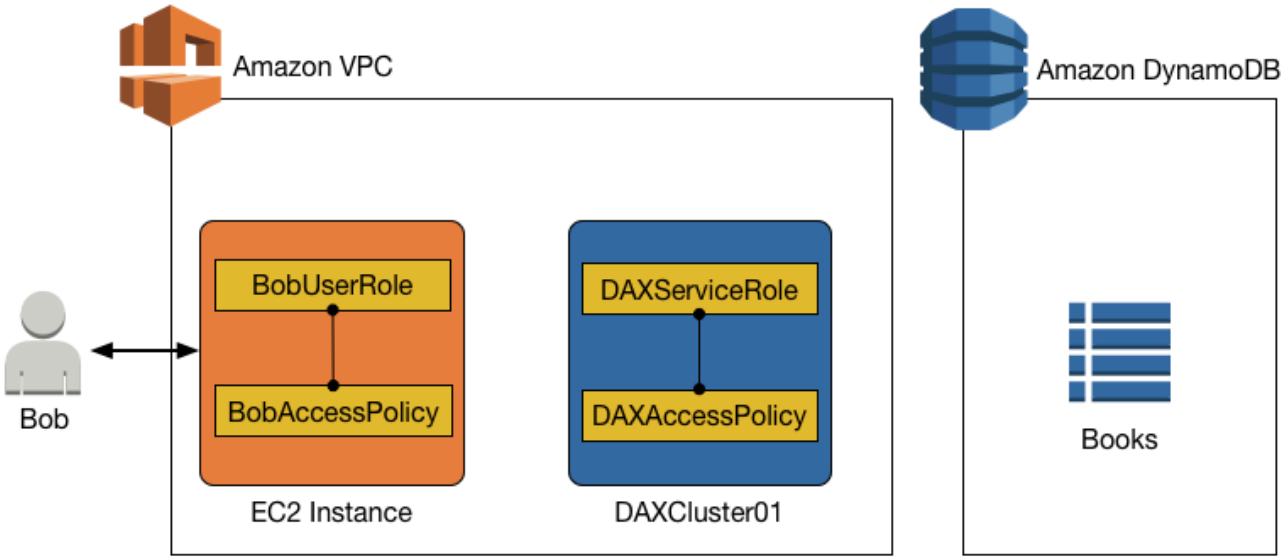
Para la usuaria Alice, primero habría que crear `AliceAccessPolicy` con el documento de política que se muestra más arriba. Posteriormente, se adjuntaría la política a Alice.

> **Note**
>
> En lugar de adjuntar la política a un usuario, podría adjuntarla a un rol de IAM. De ese modo, todos los usuarios que asuman ese rol tendrían los permisos definidos en la política.

La política de usuario, junto con la función del servicio de DAX, determinan los recursos de DynamoDB y las acciones de la API a los que el destinatario puede obtener acceso a través de DAX.

Caso práctico: acceso a DynamoDB y DAX

El siguiente escenario puede ayudarle a entender mejor sobre las políticas de IAM para usarlas con DAX. (Durante el resto de esta sección haremos referencia a este escenario). En el siguiente diagrama se muestra información general sobre el escenario.



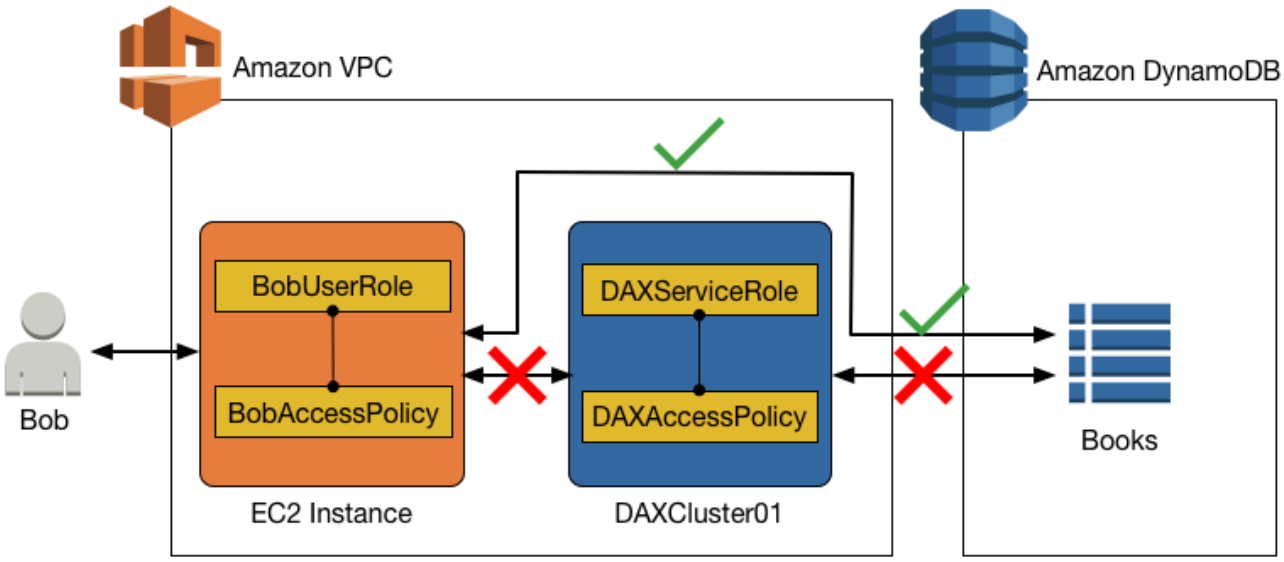
En este escenario, existen las siguientes entidades:

- Un usuario (Bob).
- Un rol de IAM (`BobUserRole`). Bob asume este rol en tiempo de ejecución.
- Una política de IAM (`BobAccessPolicy`). Esta política se asocia a `BobUserRole`. `BobAccessPolicy` define los recursos de DynamoDB y DAX a los que `BobUserRole` está autorizado a acceder.
- Un clúster DAX (`DAXCluster01`).
- Una función del servicio de IAM (`DAXServiceRole`). Este rol le permite a `DAXCluster01` acceder a DynamoDB.

- Una política de IAM (`DAXAccessPolicy`). Esta política se asocia a `DAXServiceRole`. `DAXAccessPolicy` define los recursos y las API de DynamoDB a los que `DAXCluster01` está autorizado a acceder.
- Una tabla de DynamoDB (`Books`).

La combinación de instrucciones de política en `BobAccessPolicy` y `DAXAccessPolicy` determina lo que Bob puede hacer con la tabla `Books`. Por ejemplo, Bob podría acceder a `Books` directamente (a través del punto de enlace de DynamoDB), indirectamente (mediante el clúster de DAX) o de las dos maneras. Además, Bob podría ser capaz de leer datos de `Books`, de escribir datos en `Books` o de ambas cosas.

Acceso a DynamoDB, pero no con DAX



Es posible permitir el acceso directo a una tabla de DynamoDB, pero impedir el acceso indirecto a través de un clúster de DAX. Para el acceso directo a DynamoDB, los permisos para `BobUserRole` están determinados por la `BobAccessPolicy` (que se asocia al rol).

Acceso de solo lectura a DynamoDB (solamente)

Bob puede acceder a DynamoDB con `BobUserRole`. La política de IAM asociada a este rol (`BobAccessPolicy`) determina las tablas de DynamoDB a las que `BobUserRole` puede acceder y a las API que `BobUserRole` puede invocar.

Supongamos que vamos a usar el documento de política siguiente para `BobAccessPolicy`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Cuando este documento se asocie a `BobAccessPolicy`, permitirá que `BobUserRole` obtenga acceso al punto de enlace de DynamoDB y lleve a cabo operaciones de solo lectura en la tabla `Books`.

DAX no aparece en esta política, por lo que acceder a través de DAX está denegado.

Acceso de lectura y escritura a DynamoDB (solamente)

Si `BobUserRole` requiere acceso de lectura y escritura a DynamoDB, la siguiente política lo permitiría:

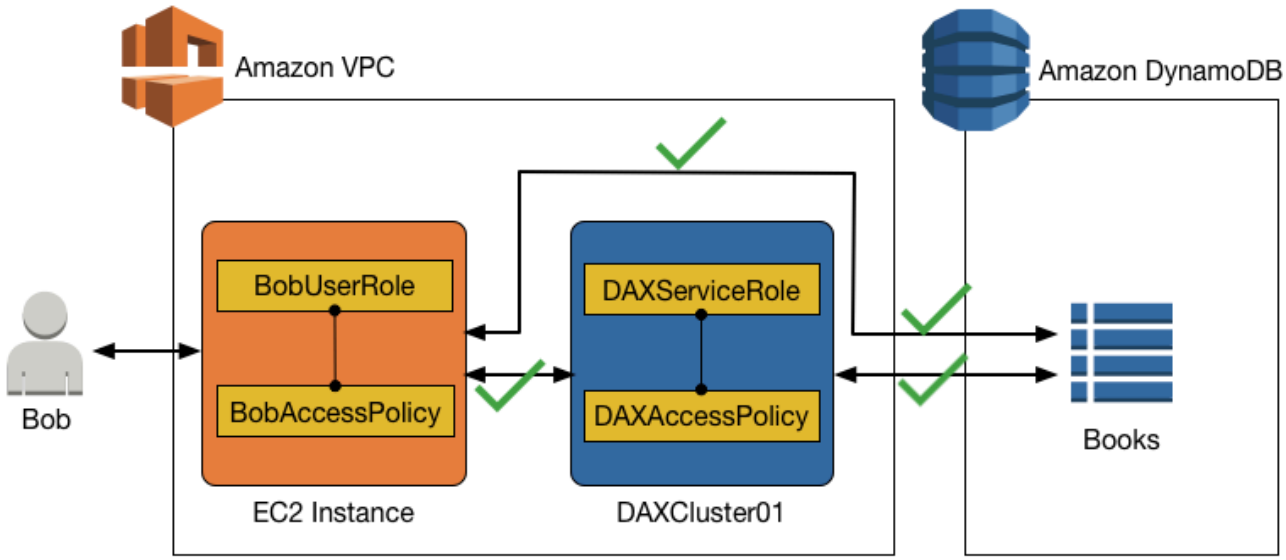
JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
```

```
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Tampoco en este caso aparece DAX en la política, por lo que el acceso a través de DAX está denegado.

Acceder a DynamoDB y DAX



Para permitir el acceso a un clúster de DAX, debe incluir las acciones específicas de DAX en una política de IAM.

Las siguientes acciones específicas de DAX se corresponden con sus homólogas de nombres parecidos de la API de DynamoDB:

- `dax:GetItem`
- `dax:BatchGetItem`
- `dax:Query`
- `dax:Scan`
- `dax:PutItem`
- `dax:UpdateItem`
- `dax:DeleteItem`
- `dax:BatchWriteItem`
- `dax:ConditionCheckItem`

Lo mismo sucede para la clave de condición `dax:EnclosingOperation`.

Acceso de solo lectura a DynamoDB y acceso de solo lectura a DAX

Supongamos que Bob requiere acceso de solo lectura a la tabla `Books` tanto desde DynamoDB como desde DAX. La siguiente política (asociada a `BobUserRole`) concedería este acceso:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
```

```
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

La política contiene una instrucción para el acceso a DAX (`DAXAccessStmt`) y otra para el acceso a DynamoDB (`DynamoDBAccessStmt`). Estas instrucciones permitirían que Bob envíe solicitudes `GetItem`, `BatchGetItem`, `Query` y `Scan` a `DAXCluster01`.

Sin embargo, la función del servicio de `DAXCluster01` también requeriría acceso de solo lectura a la tabla `Books` en DynamoDB. La siguiente política de IAM, asociada a `DAXServiceRole`, permitiría cumplir este requisito:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Acceso de lectura y escritura a DynamoDB y acceso de solo lectura con DAX

Para un rol de usuario determinado, puede proporcionar acceso de lectura y escritura a una tabla de DynamoDB y también permitir el acceso de solo lectura a través de DAX.

En el caso de Bob, la política de IAM para `BobUserRole` tendría que permitir las acciones de lectura y escritura de DynamoDB en la tabla `Books` y, además, admitir las acciones de solo lectura a través de `DAXCluster01`.

El siguiente ejemplo de documento de política para `BobUserRole` concedería este acceso:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
```

```
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Además, `DAXServiceRole` requeriría una política de IAM que permitiese que `DAXCluster01` realizase acciones de solo lectura en la tabla `Books`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:DescribeTable"
           ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Acceso de lectura y escritura a DynamoDB y acceso de lectura y escritura a DAX

Ahora, supongamos que Bob requiere acceso de lectura y escritura a la tabla `Books`, ya sea directamente desde DynamoDB o indirectamente desde `DAXCluster01`. La siguiente política, asociada a `BobAccessPolicy`, concedería este acceso:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        },
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

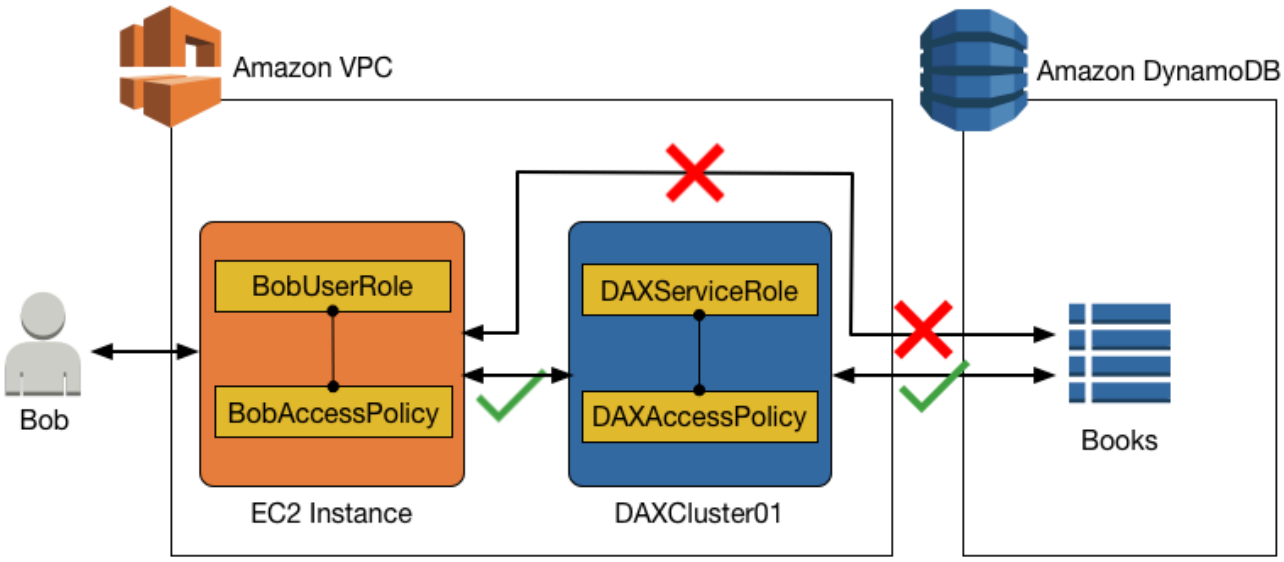
Además, `DAXServiceRole` requeriría una política de IAM que permitiese que `DAXCluster01` realizase acciones de lectura y escritura en la tabla `Books`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Acceso a DynamoDB a través de DAX, pero sin acceso directo a DynamoDB

En este caso, Bob puede obtener acceso a la tabla `Books` a través de DAX, pero no tiene acceso directo a la tabla `Books` en DynamoDB. Por lo tanto, cuando Bob recibe acceso a DAX, también recibe acceso a una tabla de DynamoDB a la que, de otro modo, no podría obtener acceso. Al configurar una política de IAM para la función del servicio de DAX, recuerde que cualquier usuario a quien se conceda acceso al clúster de DAX mediante la política de acceso de usuario recibe también acceso a las tablas especificadas en esa política. En este caso, `BobAccessPolicy` obtiene acceso a las tablas especificadas en `DAXAccessPolicy`.



Si está utilizando los roles y las políticas de IAM para restringir el acceso a los datos y las tablas de DynamoDB, el uso de DAX puede alterar las políticas. En la política siguiente, Bob tiene acceso a una tabla de DynamoDB a través de DAX, aunque no tiene acceso explícito directo a esa misma tabla en DynamoDB.

El siguiente documento de política (`BobAccessPolicy`), asociado a `BobUserRole`, concedería este acceso:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DAXAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
```

```
                "dax:DeleteItem",
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-west-2:123456789012:cache/DAXCluster01"
        }
    ]
}
```

En esta política de acceso, no hay permisos para obtener acceso a DynamoDB directamente.

Junto con `BobAccessPolicy`, la siguiente `DAXAccessPolicy` concede acceso a `BobUserRole` a la tabla `Books` de DynamoDB incluso si `BobUserRole` no puede acceder directamente a la tabla `Books`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBAccessStmt",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:DescribeTable",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Como se muestra en este ejemplo, cuando se configura el control de acceso para la política de acceso de usuario y la política de acceso de clúster de DAX, es preciso comprender perfectamente todos los permisos de acceso que se conceden, con el fin de asegurarse de cumplir el principio de mínimos privilegios. Además, debe asegurarse de que, al conceder a un usuario acceso a un clúster de DAX, no se alteren las políticas de control de acceso establecidas previamente.

Cifrado en reposo de DAX

El cifrado de Amazon DynamoDB Accelerator (DAX) en reposo proporciona una capa adicional de protección de datos al ayudarle a proteger los datos del acceso no autorizado al almacenamiento subyacente. Las políticas de la organización, las normativas industriales o gubernamentales y los requisitos de conformidad pueden requerir el uso del cifrado en reposo para proteger los datos. Puede utilizar el cifrado para aumentar la seguridad de datos de las aplicaciones implementadas en la nube.

Con el cifrado en reposo, los datos mantenidos por DAX en disco se cifran utilizando el estándar de cifrado avanzado de 256 bits, también conocido como cifrado AES-256. DAX escribe datos en disco como parte de la propagación de cambios desde el nodo principal a las réplicas de lectura.

El cifrado en reposo de DAX se integra de forma automática con AWS Key Management Service (AWS KMS) para administrar la única clave de servicio predeterminada que se utiliza para cifrar los clústeres. Si no existe una clave de servicio predeterminada al crear el clúster de DAX cifrado, AWS KMS le crea automáticamente una nueva clave administrada por AWS. Esta clave se utilizará con los clústeres cifrados que se creen en el futuro. AWS KMS combina hardware y software seguros de alta disponibilidad para ofrecer un sistema de administración de claves adaptado a la nube.

Una vez cifrados los datos, DAX se encarga del descifrado de los datos de forma transparente con un impacto mínimo en el rendimiento. No es necesario modificar las aplicaciones para utilizar el cifrado.

> **Note**
>
> DAX no llama a AWS KMS para cada operación de DAX. DAX solo utiliza la clave durante el lanzamiento del clúster. Incluso si se revoca el acceso, DAX puede seguir accediendo a los datos hasta que se cierra el clúster. Las claves de AWS KMS especificadas por el cliente no se admiten.

El cifrado en reposo de DAX está disponible para los siguientes tipos de nodo de clúster.

Familia	Tipo de nodo
Optimizada para memoria (R4, R5 y R7)	dax.r4.large
	dax.r4.xlarge
	dax.r4.2xlarge
	dax.r4.4xlarge
	dax.r4.8xlarge
	dax.r4.16xlarge
	dax.r5.large
	dax.r5.xlarge
	dax.r5.2xlarge
	dax.r5.4xlarge
	dax.r5.8xlarge
	dax.r5.12xlarge
	dax.r5.16xlarge
	dax.r5.24xlarge
	dax.r7i.large
	dax.r7i.xlarge
	dax.r7i.2xlarge
	dax.r7i.4xlarge
	dax.r7i.8xlarge
	dax.r7i.12xlarge
	dax.r7i.16xlarge

Familia	Tipo de nodo
	dax.r7i.24xlarge
Propósito general (T2)	dax.t2.small dax.t2.medium
Propósito general (T3)	dax.t3.small dax.t3.medium

> **Important**
>
> El cifrado en reposo de DAX no se admite para tipos de nodo `dax.r3.*`.

No puede habilitar o deshabilitar el cifrado en reposo después de haber creado un clúster. Debe volver a crear el clúster para habilitar el cifrado en reposo si no se habilitó durante la creación.

El cifrado en reposo de DAX se ofrece sin coste adicional (se aplican cargos por el uso de claves de cifrado de AWS KMS). Para obtener más información sobre los precios, consulte Precios de Amazon DynamoDB.

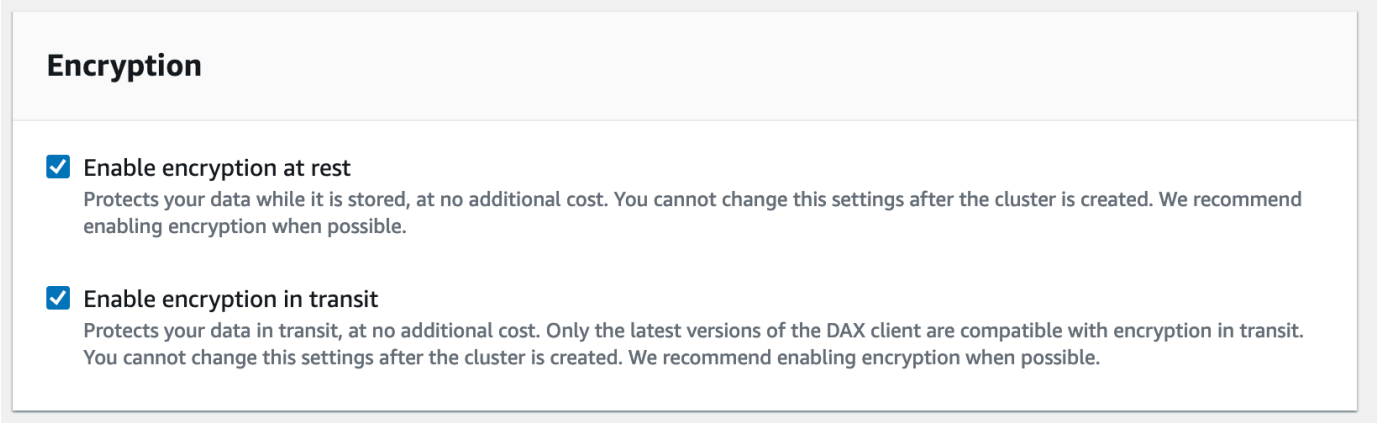
Habilitación del cifrado en reposo mediante la Consola de administración de AWS

Siga estos pasos para habilitar el cifrado en reposo de DAX en una tabla mediante la consola.

Para habilitar el cifrado en reposo de DAX

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación en el lado izquierdo de la consola, en DAX, elija Clusters (Clústeres).
3. Elija Create cluster.
4. En Cluster name (Nombre del clúster), escriba un nombre abreviado para su clúster. Elija el node type (tipo de nodo) de todos los nodos del clúster y para el tamaño del clúster, utilice **3** nodos.

5. En Encryption (Cifrado), asegúrese de que Enable encryption (Habilitar el cifrado) esté seleccionado.



6. Después de elegir el rol de IAM, grupo de subred, grupos de seguridad y configuración de clúster, elija Launch cluster (Lanzar clúster).

Para confirmar que el clúster está cifrado, compruebe los datos del clúster en el panel Clusters (Clústeres). El cifrado debe estar ENABLED (HABILITADO).

Cifrado en tránsito de DAX

El Amazon DynamoDB Accelerator (DAX) admite el cifrado en tránsito de datos entre la aplicación y el clúster DAX, lo que le permite utilizar DAX en aplicaciones con requisitos de cifrado estrictos.

Independientemente de si elige o no el cifrado en tránsito, el tráfico entre la aplicación y el clúster DAX permanece en su Amazon VPC. Este tráfico se enruta a las interfaces de red elástica con IP privadas de la VPC que están adjuntas a los nodos del clúster. Con su VPC como límite de confianza, tiene un control significativo sobre la seguridad de sus datos mediante el uso de herramientas estándar como grupos de seguridad, segmentación de subredes con ACL de red y seguimiento de flujo de VPC. El cifrado en tránsito de DAX añade a este nivel básico de confidencialidad, lo que garantiza que todas las solicitudes y respuestas entre la aplicación y el clúster estén cifradas por seguridad de nivel de transporte (TLS), y las conexiones al clúster se pueden autenticar mediante la verificación de un certificado x509 del clúster. Los datos escritos en disco por DAX también se pueden cifrar si elige el cifrado en reposo al momento de crear su clúster de DAX.

Usar el cifrado en tránsito con DAX es fácil. Simplemente seleccione esta opción al momento de crear un nuevo clúster y utilice una versión reciente de cualquiera de los clientes de DAX en su aplicación. Los clústeres que usan cifrado en tránsito no admiten tráfico no cifrado, por lo que no hay

posibilidad de configurar mal la aplicación y eludir el cifrado. El cliente DAX utilizará el certificado x509 del clúster para autenticar la identidad del clúster cuando establezca conexiones, garantizando que las solicitudes DAX vayan donde se desee. Todos los métodos de creación de clústeres DAX admiten el cifrado en tránsito: la Consola de administración de AWS, la AWS CLI, todos los SDK y la CloudFormation.

El cifrado en tránsito no se puede habilitar en un clúster DAX existente. Para utilizar el cifrado en tránsito en una aplicación DAX existente, cree un nuevo clúster con el cifrado en tránsito habilitado, mueva el tráfico de la aplicación hacia él y, a continuación, elimine el clúster anterior.

Uso de roles de IAM vinculados a servicios para DAX

Amazon DynamoDB Accelerator (DAX) utiliza roles vinculados a servicios de AWS Identity and Access Management (IAM). Un rol vinculado a un servicio es un tipo único de rol de IAM que está vinculado directamente a DAX. Los roles vinculados a servicios están predefinidos por DAX e incluyen todos los permisos que el servicio requiere para llamar a otros servicios de AWS en su nombre.

Un rol vinculado a un servicio simplifica la configuración de DAX porque ya no tendrá que agregar manualmente los permisos necesarios. DAX define los permisos de sus roles vinculados a servicios y, a menos que esté definido de otra manera, solo DAX puede asumir sus roles. Los permisos definidos incluyen la política de confianza y la política de permisos. Dicha política de permisos no se puede asociar a ninguna otra entidad de IAM.

Las funciones se pueden eliminar únicamente después de eliminar primero sus recursos relacionados. De esta forma, se protegen los recursos de DAX, ya que se evita que pueda eliminar accidentalmente permisos de acceso a los recursos.

Para obtener más información sobre otros servicios que admiten los roles vinculados a servicios, consulte Servicios de AWS que funcionan con IAM en la Guía del usuario de IAM. Busque los servicios para los que se indique Yes (Sí) en la columna Service-linked roles (Roles vinculados a servicios). Elija el vínculo Yes (Sí) para ver la documentación acerca del rol vinculado a un servicio en cuestión.

Temas

- Permisos de roles vinculados a servicios para DAX
- Creación de un rol vinculado a un servicio para DAX
- Edición de un rol vinculado a un servicio para DAX

- Eliminación de un rol vinculado a un servicio para DAX

Permisos de roles vinculados a servicios para DAX

DAX usa el rol vinculado a servicios denominado `AWSServiceRoleForDAX`. Este rol permite a DAX llamar a servicios en nombre de su clúster de DAX.

> **Important**
>
> El rol vinculado a un servicio `AWSServiceRoleForDAX` le facilita la configuración y el mantenimiento de un clúster de DAX. Sin embargo, debe conceder acceso a DynamoDB a cada clúster antes de poder usarlo. Para obtener más información, consulte Control de acceso a DAX.

El rol vinculado al servicio `AWSServiceRoleForDAX` confía en los siguientes servicios para asumir el rol:

- `dax.amazonaws.com`

La política de permisos del rol permite que DAX realice las siguientes acciones en los recursos especificados:

- Acciones en `ec2`:
  - `AuthorizeSecurityGroupIngress`
  - `CreateNetworkInterface`
  - `CreateSecurityGroup`
  - `DeleteNetworkInterface`
  - `DeleteSecurityGroup`
  - `DescribeAvailabilityZones`
  - `DescribeNetworkInterfaces`
  - `DescribeSecurityGroups`
  - `DescribeSubnets`
  - `DescribeVpcs`
  - `ModifyNetworkInterfaceAttribute`

- `RevokeSecurityGroupIngress`

Debe configurar permisos para permitir a una entidad de IAM (como un usuario, grupo o rol) crear, editar o eliminar un rol vinculado a servicios. Para obtener más información, consulte Permisos de roles vinculados a servicios en la Guía del usuario de IAM.

Para permitir a una entidad de IAM crear roles vinculados a servicios AWSServiceRoleForDAX

Añada la siguiente instrucción de política a los permisos para esa entidad de IAM.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": "*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "dax.amazonaws.com"}}
}
```

Creación de un rol vinculado a un servicio para DAX

No necesita crear manualmente un rol vinculado a servicios. Cuando crea un clúster, DAX se encarga de crearle el rol vinculado al servicio.

> **Important**
>
> Si utilizaba el servicio de DAX antes del 28 de febrero de 2018, cuando se comenzaron a admitir roles vinculados a servicios, DAX creó el rol `AWSServiceRoleForDAX` en su cuenta. Para obtener más información, consulte Un nuevo rol ha aparecido en mi cuenta de AWS en la Guía del usuario de IAM.

Si elimina este rol vinculado a un servicio y necesita crearlo de nuevo, puede utilizar el mismo proceso para volver a crear el rol en su cuenta. Cuando crea una instancia o un clúster, DAX se encarga de crearle de nuevo el rol vinculado al servicio.

Edición de un rol vinculado a un servicio para DAX

DAX no le permite editar el rol vinculado a servicios `AWSServiceRoleForDAX`. Después de crear un rol vinculado al servicio, no podrá cambiar el nombre del rol, ya que varias entidades podrían

hacer referencia al rol. Sin embargo, sí puede editar la descripción del rol con IAM. Para obtener más información, consulte Editar un rol vinculado a servicios en la Guía del usuario de IAM.

Eliminación de un rol vinculado a un servicio para DAX

Si ya no necesita usar una característica o servicio que requieran un rol vinculado a un servicio, le recomendamos que elimine dicho rol. De esta forma no tiene una entidad no utilizada que no se monitoree ni mantenga de forma activa. Sin embargo, debe eliminar todos los clústeres de DAX para poder eliminar el rol vinculado al servicio.

Limpiar un rol vinculado a servicios

Antes de poder utilizar IAM para eliminar un rol vinculado a un servicio, primero debe confirmar que dicho rol no tiene sesiones activas y eliminar los recursos que utiliza.

Para comprobar si el rol vinculado a un servicio tiene una sesión activa en la consola de IAM

1. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en https://console.aws.amazon.com/iam/.
2. En el panel de navegación de la consola de IAM, elija Roles. Luego, elija el nombre (no la casilla de verificación) del rol AWSServiceRoleForDAX.
3. En la página Resumen del rol seleccionado, seleccione la pestaña Asesor de acceso.
4. En la pestaña Asesor de acceso, revise la actividad reciente del rol vinculado a servicios.

> **Note**
>
> Si no está seguro de si DAX utiliza el rol `AWSServiceRoleForDAX`, puede intentar eliminar el rol para comprobarlo. Si el servicio está utilizando el rol, este no podrá eliminarse, y podrá ver las regiones en las que se está utilizando. Si el rol se está utilizando, entonces debe eliminar sus clústeres de DAX antes de poder eliminar el rol. No se puede revocar la sesión de un rol vinculado a un servicio.

Si desea eliminar el rol `AWSServiceRoleForDAX`, primero debe eliminar todos sus clústeres de DAX.

Eliminación de todos los clústeres de DAX

Use alguno de estos procedimientos para eliminar cada uno de sus clústeres de DAX.

Para eliminar un clúster de DAX (consola)

1. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación, en DAX, elija Clusters (Clústeres).
3. Elija Acciones y, a continuación, elija Eliminar.
4. En el recuadro Delete cluster confirmation (Eliminar confirmación de clúster), elija Delete (Eliminar).

Para eliminar un clúster de DAX (AWS CLI)

Consulte delete-cluster en la Referencia de comandos de la AWS CLI.

Para eliminar un clúster de DAX (API)

Consulte DeleteCluster en la Referencia de la API de Amazon DynamoDB.

Eliminación del rol vinculado a un servicio

Eliminación manual del rol vinculado a servicios mediante IAM

Utilice la consola de IAM, la CLI de IAM o la API de IAM para eliminar el rol vinculado a servicios `AWSServiceRoleForDAX`. Para más información, consulte Eliminación de un rol vinculado a servicios en la Guía del usuario de IAM.

Acceso a DAX a través de las cuentas de AWS

Imagine que tiene un clúster de DynamoDB Accelerator (DAX) ejecutándose en una cuenta de AWS (cuenta A) y que el clúster de DAX debe ser accesible desde una instancia de Amazon Elastic Compute Cloud (Amazon EC2) en otra cuenta de AWS (cuenta B). En este tutorial, lo logrará lanzando una instancia EC2 en la cuenta B con un rol de IAM de la cuenta B. A continuación, utiliza las credenciales de seguridad temporales de la instancia EC2 para asumir un rol de IAM de la cuenta A. Por último, utiliza las credenciales de seguridad temporales de asumir el rol de IAM en la cuenta A para realizar llamadas a la aplicación a través de una interconexión de Amazon VPC al clúster de DAX en la cuenta A. Para realizar estas tareas necesitará acceso administrativo en ambas cuentas de AWS.

> **Important**
>
> No es posible hacer que un clúster DAX acceda a una tabla de DynamoDB desde una cuenta diferente.

Temas

- Configurar IAM
- Configurar una VPC
- Modificar el cliente de DAX para permitir el acceso entre cuentas

Configurar IAM

1. Cree un archivo de texto denominado `AssumeDaxRoleTrust.json` con el siguiente contenido, que permite a Amazon EC2 trabajar en su nombre.

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Principal": {
                   "Service": "ec2.amazonaws.com"
               },
               "Action": "sts:AssumeRole"
           }
       ]
   }
   ```

2. En la cuenta B, cree un rol que Amazon EC2 pueda utilizar al lanzar instancias.

   ```
   aws iam create-role \
       --role-name AssumeDaxRole \
       --assume-role-policy-document file://AssumeDaxRoleTrust.json
   ```

3. Cree un archivo de texto denominado `AssumeDaxRolePolicy.json` con el siguiente contenido, que permite que el código que se ejecuta en la instancia EC2 en la cuenta B asuma un rol de IAM en la cuenta A. Reemplace *accountA* por el ID real de la cuenta A.

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": "sts:AssumeRole",
               "Resource": "arn:aws:iam::111122223333:role/DaxCrossAccountRole"
           }
       ]
   }
   ```

4. Agregue esa política al rol que acaba de crear.

   ```
   aws iam put-role-policy \
       --role-name AssumeDaxRole \
       --policy-name AssumeDaxRolePolicy \
       --policy-document file://AssumeDaxRolePolicy.json
   ```

5. Cree un perfil de instancia para permitir que las instancias utilicen el rol.

   ```
   aws iam create-instance-profile \
       --instance-profile-name AssumeDaxInstanceProfile
   ```

6. Asocie el rol con el perfil de instancia.

   ```
   aws iam add-role-to-instance-profile \
       --instance-profile-name AssumeDaxInstanceProfile \
       --role-name AssumeDaxRole
   ```

7. Cree un archivo de texto denominado `DaxCrossAccountRoleTrust.json` con el siguiente contenido, lo que permite a la cuenta B asumir un rol de cuenta A. Reemplace *cuentaB* por el ID real de la cuenta B.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/AssumeDaxRole"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

8. En la cuenta A, cree el rol que la cuenta B puede asumir.

```
aws iam create-role \
    --role-name DaxCrossAccountRole \
    --assume-role-policy-document file://DaxCrossAccountRoleTrust.json
```

9. Cree un archivo de texto denominado `DaxCrossAccountPolicy.json` que permita el acceso al clúster de DAX. Reemplace *`dax-cluster-arn`* por el nombre de recurso de Amazon (ARN) correcto de su clúster de DAX.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dax:GetItem",
                "dax:BatchGetItem",
                "dax:Query",
                "dax:Scan",
                "dax:PutItem",
                "dax:UpdateItem",
                "dax:DeleteItem",
```

```
                "dax:BatchWriteItem",
                "dax:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dax:us-east-1:111122223333:cache/dax-
cluster-name"
        }
    ]
}
```

10. En la cuenta A, agregue la política al rol.

```
aws iam put-role-policy \
    --role-name DaxCrossAccountRole \
    --policy-name DaxCrossAccountPolicy \
    --policy-document file://DaxCrossAccountPolicy.json
```

Configurar una VPC

1. Busque el grupo de subred del clúster de DAX de la cuenta A. Reemplace *cluster-name* por el nombre del clúster de DAX al que debe tener acceso la cuenta B.

```
aws dax describe-clusters \
    --cluster-name cluster-name
    --query 'Clusters[0].SubnetGroup'
```

2. Utilizando ese *grupo de subred*, busque la VPC del clúster.

```
aws dax describe-subnet-groups \
    --subnet-group-name subnet-group \
    --query 'SubnetGroups[0].VpcId'
```

3. Usando ese *vpc-id*, busque el CIDR de la VPC.

```
aws ec2 describe-vpcs \
    --vpc vpc-id \
    --query 'Vpcs[0].CidrBlock'
```

4. Desde la cuenta B, cree una VPC utilizando un CIDR diferente y no superpuesto que el encontrado en el paso anterior. A continuación, cree al menos una subred. Puede utilizar el asistente de creación de VPC en la Consola de administración de AWS o en la AWS CLI.

5. Desde la cuenta B, solicite una interconexión a la VPC de la cuenta A como se describe en Creación y aceptación de una conexión de emparejamiento de VPC. Desde la cuenta A, acepte la conexión.
6. En la cuenta B, busque la nueva tabla de ruteo de la VPC. Reemplace *vpc-id* por el ID de la VPC que creó en la cuenta B.

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id,Values=vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

7. Agregue una ruta para enviar tráfico destinado al CIDR de la cuenta A a la interconexión de VPC. Recuerde reemplazar cada *marcador de posición de entrada de usuario* por los valores correctos para sus cuentas.

```
aws ec2 create-route \
    --route-table-id accountB-route-table-id \
    --destination-cidr accountA-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

8. En la cuenta A, busque la tabla de enrutamiento del clúster de DAX utilizando el *vpc-id* que encontró anteriormente.

```
aws ec2 describe-route-tables \
    --filters 'Name=vpc-id, Values=accountA-vpc-id' \
    --query 'RouteTables[0].RouteTableId'
```

9. Desde la cuenta A, agregue una ruta para enviar tráfico destinado al CIDR de la cuenta B a la interconexión de VPC. Reemplace cada *marcador de posición de entrada de usuario* por los valores correctos para sus cuentas.

```
aws ec2 create-route \
    --route-table-id accountA-route-table-id \
    --destination-cidr accountB-vpc-cidr \
    --vpc-peering-connection-id peering-connection-id
```

10. Desde la cuenta B, lance una instancia EC2 en la VPC que creó anteriormente. Dele el `AssumeDaxInstanceProfile`. Puede utilizar el asistente de inicio en la Consola de administración de AWS o en la AWS CLI. Tome nota del grupo de seguridad de la instancia.
11. En la cuenta A, busque el grupo de seguridad utilizado por el clúster de DAX. Recuerde sustituir *cluster-name* por el nombre de su clúster de DAX.

```
aws dax describe-clusters \
    --cluster-name cluster-name \
    --query 'Clusters[0].SecurityGroups[0].SecurityGroupIdentifier'
```

12. Actualice el grupo de seguridad del clúster de DAX para permitir el tráfico entrante desde el grupo de seguridad de la instancia EC2 que creó en la cuenta B. Recuerde sustituir los *marcadores de posición de entrada del usuario* por los valores correctos para sus cuentas.

```
aws ec2 authorize-security-group-ingress \
    --group-id accountA-security-group-id \
    --protocol tcp \
    --port 8111 \
    --source-group accountB-security-group-id \
    --group-owner accountB-id
```

En este punto, una aplicación en la instancia EC2 de la cuenta B puede utilizar el perfil de instancias para asumir el rol `arn:aws:iam::accountA-id:role/DaxCrossAccountRole` y utilizar el clúster de DAX.

Modificar el cliente de DAX para permitir el acceso entre cuentas

> **Note**
>
> Las credenciales de AWS Security Token Service (AWS STS) son credenciales temporales. Algunos clientes manejan la actualización automáticamente, mientras que otros requieren lógica adicional para actualizar las credenciales. Le recomendamos que siga las instrucciones de la documentación correspondiente.

Java

Esta sección le ayuda a modificar el código de cliente de DAX existente para permitir el acceso de DAX entre cuentas. Si aún no tiene código de cliente de DAX, puede encontrar ejemplos de código de trabajo en el tutorial Java y DAX.

1. Agregue las siguientes importaciones:

```
import com.amazonaws.auth.STSAssumeRoleSessionCredentialsProvider;
import com.amazonaws.services.securitytoken.AWSSecurityTokenService;
import
 com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
```

2. Obtenga un proveedor de credenciales de AWS STS y cree un objeto cliente de DAX. Recuerde reemplazar cada *marcador de posición de entrada de usuario* por los valores correctos para sus cuentas.

```
AWSSecurityTokenService awsSecurityTokenService =
 AWSSecurityTokenServiceClientBuilder
     .standard()
     .withRegion(region)
     .build();

STSAssumeRoleSessionCredentialsProvider credentials =  new
 STSAssumeRoleSessionCredentialsProvider.Builder("arn:aws:iam::accountA:role/
RoleName", "TryDax")
     .withStsClient(awsSecurityTokenService)
     .build();

DynamoDB client = AmazonDaxClientBuilder.standard()
    .withRegion(region)
    .withEndpointConfiguration(dax_endpoint)
    .withCredentials(credentials)
    .build();
```

.NET

Esta sección le ayuda a modificar el código de cliente de DAX existente para permitir el acceso de DAX entre cuentas. Si aún no tiene código de cliente de DAX, puede encontrar ejemplos de código de trabajo en el tutorial .NET y DAX.

1. Agregue el paquete NuGet AWSSDK.SecurityToken a la solución.

```
<PackageReference Include="AWSSDK.SecurityToken" Version="latest version" />
```

2. Utilice los paquetes `SecurityToken` y `SecurityToken.Model`.

```
using Amazon.SecurityToken;
```

```
using Amazon.SecurityToken.Model;
```

3. Obtenga credenciales temporales de `AmazonSimpleTokenService` y cree un objeto de `ClusterDaxClient`. Recuerde reemplazar cada *`marcador de posición de entrada de usuario`* por los valores correctos para sus cuentas.

```
IAmazonSecurityTokenService sts = new AmazonSecurityTokenServiceClient();

var assumeRoleResponse = sts.AssumeRole(new AssumeRoleRequest
{
    RoleArn = "arn:aws:iam::accountA:role/RoleName",
                RoleSessionName = "TryDax"
});

Credentials credentials = assumeRoleResponse.Credentials;

var clientConfig = new DaxClientConfig(dax_endpoint, port)
{
    AwsCredentials = assumeRoleResponse.Credentials

};

var client = new ClusterDaxClient(clientConfig);
```

Go

Esta sección le ayuda a modificar el código de cliente de DAX existente para permitir el acceso de DAX entre cuentas. Si aún no tiene código de cliente de DAX, puede encontrar ejemplos de código en funcionamiento en GitHub.

1. Importe los paquetes de AWS STS y de sesión.

```
import (
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/sts"
    "github.com/aws/aws-sdk-go/aws/credentials/stscreds"
)
```

2. Obtener credenciales temporales de `AmazonSimpleTokenService` y crear un objeto cliente de DAX. Recuerde reemplazar cada *`marcador de posición de entrada de usuario`* por los valores correctos para sus cuentas.

```
sess, err := session.NewSession(&aws.Config{
    Region: aws.String(region)},
)
if err != nil {
    return nil, err
}

stsClient := sts.New(sess)
arp := &stscreds.AssumeRoleProvider{
                Duration:     900 * time.Second,
                ExpiryWindow: 10 * time.Second,
                RoleARN:      "arn:aws:iam::accountA:role/role_name",
                Client:       stsClient,
                RoleSessionName: "session_name",
        }cfg := dax.DefaultConfig()

cfg.HostPorts = []string{dax_endpoint}
cfg.Region = region
cfg.Credentials = credentials.NewCredentials(arp)
daxClient := dax.New(cfg)
```

Python

Esta sección le ayuda a modificar el código de cliente de DAX existente para permitir el acceso de DAX entre cuentas. Si aún no tiene código de cliente de DAX, puede encontrar ejemplos de código de trabajo en el tutorial Python y DAX.

1. Importe `boto3`.

```
import boto3
```

2. Obtenga credenciales temporales de `sts` y cree un objeto `AmazonDaxClient`. Recuerde reemplazar cada *`marcador de posición de entrada de usuario`* por los valores correctos para sus cuentas.

```
sts = boto3.client('sts')
stsresponse =
 sts.assume_role(RoleArn='arn:aws:iam::accountA:role/
RoleName',RoleSessionName='tryDax')
credentials = botocore.session.get_session()['Credentials']
```

```
dax = amazondax.AmazonDaxClient(session, region_name=region,
 endpoints=[dax_endpoint], aws_access_key_id=credentials['AccessKeyId'],
 aws_secret_access_key=credentials['SecretAccessKey'],
 aws_session_token=credentials['SessionToken'])
client = dax
```

Node.js

Esta sección le ayuda a modificar el código de cliente de DAX existente para permitir el acceso de DAX entre cuentas. Si aún no tiene código de cliente de DAX, puede encontrar ejemplos de código de trabajo en el tutorial Node.js y DAX. Recuerde reemplazar cada *marcador de posición de entrada de usuario* por los valores correctos para sus cuentas.

```
const AmazonDaxClient = require('amazon-dax-client');
const AWS = require('aws-sdk');
const region = 'region';
const endpoints = [daxEndpoint1, ...];

const getCredentials = async() => {
  return new Promise((resolve, reject) => {
    const sts = new AWS.STS();
    const roleParams = {
      RoleArn: 'arn:aws:iam::accountA:role/RoleName',
      RoleSessionName: 'tryDax',
    };
    sts.assumeRole(roleParams, (err, session) => {
      if(err) {
        reject(err);
      } else {
        resolve({
          accessKeyId: session.Credentials.AccessKeyId,
          secretAccessKey: session.Credentials.SecretAccessKey,
          sessionToken: session.Credentials.SessionToken,
        });
      }
    });
  });
};

const createDaxClient = async() => {
  const credentials = await getCredentials();
```

```
  const daxClient = new AmazonDaxClient({endpoints: endpoints, region: region,
 accessKeyId: credentials.accessKeyId, secretAccessKey: credentials.secretAccessKey,
 sessionToken: credentials.sessionToken});
  return new AWS.DynamoDB.DocumentClient({service: daxClient});
};

createDaxClient().then((client) => {
  client.get(...);
  ...
}).catch((error) => {
  console.log('Caught an error: ' + error);
});
```

Guía de tamaño del clúster de DAX

Esta guía proporciona consejos para elegir un tamaño de clúster de Amazon DynamoDB Accelerator (DAX) y un tipo de nodo adecuados para su aplicación. Estas instrucciones le guiarán a través de los pasos necesarios para estimar el tráfico de DAX de su aplicación, seleccionar una configuración de clúster y probarla.

Si tiene un clúster de DAX existente y desea evaluar si tiene el número y el tamaño de nodos adecuados, consulte Escalado de un clúster de DAX.

Temas

- Información general
- Estimación del tráfico
- Prueba de carga

Información general

Es importante escalar el clúster de DAX de forma adecuada para su carga de trabajo, ya sea que esté creando un clúster nuevo o manteniendo un clúster existente. A medida que pasa el tiempo y cambia la carga de trabajo de la aplicación, debe revisar periódicamente las decisiones de escalado para asegurarse de que sigan siendo apropiadas.

El proceso suele seguir estos pasos:

1. Estimación del tráfico. En este paso, realiza predicciones sobre el volumen de tráfico al que enviará la aplicación a DAX, la naturaleza del tráfico (operaciones de lectura o de escritura) y la tasa de aciertos de caché esperada.
2. Prueba de carga. En este paso, crea un clúster y le envía tráfico reflejando las estimaciones del paso anterior. Repita este paso hasta que encuentre una configuración de clúster adecuada.
3. Monitoreo de la producción. Mientras la aplicación está utilizando DAX en producción, debe monitorear el clúster para validar continuamente que aún se ha escalado correctamente a medida que la carga de trabajo cambia con el tiempo.

Estimación del tráfico

Hay tres factores principales que caracterizan una carga de trabajo típica de DAX:

- Tasa de aciertos de caché
- Unidades de capacidad de lectura (RCU) por segundo
- Unidades de capacidad de escritura (WCU) por segundo

Estimación de la tasa de aciertos de caché

Si ya tiene un clúster de DAX, puede utilizar `ItemCacheHits` y `ItemCacheMisses` Métricas de Amazon CloudWatch para determinar la tasa de aciertos de caché. La tasa de aciertos de caché es igual a `ItemCacheHits` / (`ItemCacheHits` + `ItemCacheMisses`). Si la carga de trabajo incluye operaciones `Query` o `Scan`, también debe consultar las métricas `QueryCacheHits`, `QueryCacheMisses`, `ScanCacheHits` y `ScanCacheMisses`. La tasa de aciertos de caché varía de una aplicación a otra y están muy influenciadas por la configuración del período de vida (TTL) del clúster. Las tasas de acierto típicas para las aplicaciones que utilizan DAX son de entre el 85 al 95 por ciento.

Estimación de unidades de capacidad de lectura y escritura

Si ya tiene tablas de DynamoDB para su aplicación, vea `ConsumedReadCapacityUnits` y `ConsumedWriteCapacityUnits` Métricas de CloudWatch. Utilice la estadística `Sum` y divida por el número de segundos del período.

Si también tiene un clúster de DAX, recuerde que la métrica `ConsumedReadCapacityUnits` de DynamoDB sólo tiene en cuenta los errores de caché. Por lo tanto, para tener una idea de las

unidades de capacidad de lectura por segundo manejadas por su clúster de DAX, divida el número por su tasa de errores de caché (es decir, 1 - tasa de aciertos de caché).

Si aún no dispone de una tabla de DynamoDB, consulte la documentación sobre las unidades de capacidad de lectura para calcular su tráfico en función de la tasa de solicitudes estimada de su aplicación, los elementos a los que se accede por solicitud y el tamaño de los elementos.

Al realizar estimaciones de tráfico, planifique el crecimiento futuro y los picos esperados e inesperados para asegurarse de que el clúster tenga suficiente margen de ampliación para los aumentos de tráfico.

Prueba de carga

El siguiente paso después de estimar el tráfico consiste en probar la configuración del clúster bajo carga.

1. Para la prueba de carga inicial, le recomendamos que comience con el tipo de nodo `dax.r4.large`, el rendimiento fijo de menor costo y el tipo de nodo optimizado para memoria.
2. Un clúster tolerante a errores requiere al menos tres nodos, distribuidos en tres zonas de disponibilidad. En este caso, si una zona de disponibilidad deja de estar disponible, el número efectivo de zonas de disponibilidad se reduce en un tercio. Para la prueba de carga inicial, se recomienda comenzar con un clúster de dos nodos, que simula el error de una zona de disponibilidad en un clúster de tres nodos.
3. Dirija tráfico sostenido (como se estimó en el paso anterior) al clúster de prueba durante la duración de la prueba de carga.
4. Monitoree el rendimiento del clúster durante la prueba de carga.

Idealmente, el perfil de tráfico que dirija durante la prueba de carga debería ser lo más similar posible al tráfico real de la aplicación. Esto incluye la distribución de operaciones (por ejemplo, 70 por ciento `GetItem`, 25 por ciento `Query`, y 5 por ciento `PutItem`), la tasa de solicitudes para cada operación, el número de elementos a los que se accede por solicitud y la distribución de tamaños de elementos. Para lograr una tasa de aciertos de caché similar a la tasa de aciertos de caché esperada de su aplicación, preste atención a la distribución de claves en el tráfico de prueba.

> **Note**
>
> Tenga cuidado al probar los tipos de nodos T2 (`dax.t2.small` y `dax.t2.medium`). Los tipos de nodos T2 proporcionan un rendimiento de CPU con ráfagas que varía con el tiempo

en función del saldo de crédito de CPU del nodo. Un clúster de DAX que se ejecuta en nodos T2 puede parecer que funciona con normalidad, pero si algún nodo se amplió por encima del rendimiento de la línea de base de su instancia, el nodo está gastando su saldo de crédito acumulado de CPU. Cuando el saldo de crédito es bajo, el rendimiento se reduce gradualmente al nivel de rendimiento de referencia.

Monitoree el clúster de DAX durante la prueba de carga para determinar si el tipo de nodo que está utilizando para la prueba de carga es el tipo de nodo adecuado para usted. Además, durante una prueba de carga, debe monitorear la tasa de solicitudes y la tasa de aciertos de caché para asegurarse de que su infraestructura de prueba realmente está dirigiendo la cantidad de tráfico que desea.

Debe prestar atención al consumo de bytes de red del tipo de instancia de clúster seleccionado. Si se supera el ancho de banda de referencia disponible para una instancia de Amazon EC2, es posible que su clúster no pueda mantener la carga de trabajo de su aplicación y sea necesario escalarlo.

Si las pruebas de carga indican que la configuración del clúster seleccionada no puede soportar la carga de trabajo de su aplicación, debe cambiar a un tipo de nodo más grande, especialmente si observa un uso de CPU elevado en el nodo principal del clúster, altas tasas de expulsión o un uso alto de la memoria de caché. Si las tasas de aciertos son altas de forma coherente y la proporción de tráfico de lectura a escritura es alta, es posible que desee plantearse la posibilidad de agregar más nodos al clúster. Consulte Escalado de un clúster de DAX para obtener información adicional sobre cuándo utilizar un tipo de nodo mayor (escalado vertical) o agregar más nodos (escalado horizontal).

Debe repetir la prueba de carga después de realizar cambios en la configuración del clúster.

Modelado de datos para tablas de DynamoDB

Antes de profundizar en el modelado de datos, es importante comprender algunos fundamentos de DynamoDB. DynamoDB es una base de datos NoSQL de clave-valor que permite esquemas flexibles. El conjunto de atributos de los datos, aparte de los atributos clave de cada elemento, puede ser uniforme o discreto. El esquema de claves de DynamoDB tiene la forma de una clave principal simple en la que una clave de partición identifica de forma exclusiva un elemento o la forma de una clave primaria compuesta en la que una combinación de una clave de partición y una clave de clasificación define de forma exclusiva un elemento. Se crea un hash de la clave de partición para determinar la ubicación física de los datos y recuperarlos. Por lo tanto, es importante elegir un atributo de alta cardinalidad y escalable horizontalmente como clave de partición para garantizar una distribución uniforme de los datos. El atributo de clave de clasificación es opcional en el esquema de claves y disponer de una clave de clasificación permite modelar relaciones de uno a varios y crear colecciones de elementos en DynamoDB. Además, las claves de clasificación también se denominan claves de intervalo: se utilizan para clasificar elementos de una colección de elementos y también permiten realizar operaciones flexibles basadas en intervalos.

Para obtener más detalles y las prácticas recomendadas acerca del esquema de claves de DynamoDB, puede consultar lo siguiente:

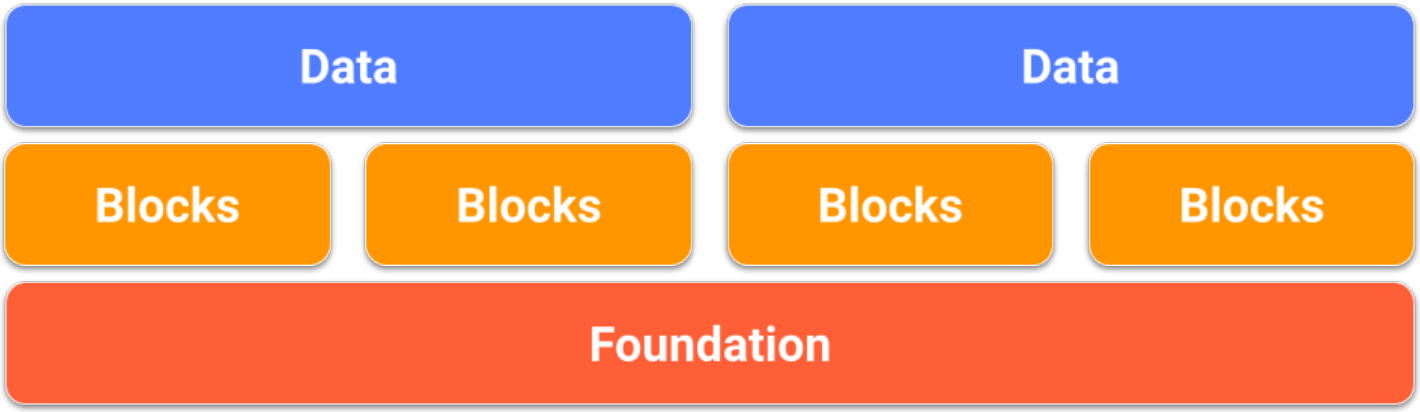
- the section called “Particiones y distribución de datos en DynamoDB”
- the section called “Diseño de claves de partición”
- the section called “Diseño de la clave de clasificación”
- Elección de la clave de partición de DynamoDB adecuada

Los índices secundarios suelen ser necesarios para admitir patrones de consulta adicionales en DynamoDB. Los índices secundarios son tablas sombra en las que los mismos datos se organizan mediante un esquema de claves distinto al de la tabla base. Un índice secundario local (LSI) comparte la misma clave de partición que la tabla base y permite tener una clave de clasificación alternativa con la que puede compartir la capacidad de la tabla base. Un índice secundario global (GSI) puede tener una clave de partición y un atributo de clave de clasificación diferentes a los de la tabla base, lo que significa que la administración del rendimiento de un GSI es independiente de la tabla base.

Para obtener más detalles sobre los índices secundarios y las prácticas recomendadas, puede consultar lo siguiente:

- the section called “Uso de índices”
- the section called “Índices secundarios”

Analicemos ahora el modelado de datos un poco más. El proceso de diseñar un esquema flexible y altamente optimizado en DynamoDB, o en cualquier base de datos NoSQL, puede ser una habilidad difícil de aprender. El objetivo de este módulo es ayudarlo a desarrollar un diagrama de flujo mental para diseñar un esquema que lo lleve del caso de uso a la producción. Empezaremos con una introducción a la elección fundamental de cualquier diseño: diseño de tabla única frente a diseño de tabla múltiple. A continuación, revisaremos la multitud de patrones de diseño (componentes) que se pueden utilizar para lograr diversos resultados organizativos o de rendimiento para la aplicación. Por último, incluimos una variedad de paquetes completos de diseño de esquemas para diferentes casos de uso y sectores.



Temas

- Colecciones de elementos: cómo modelar relaciones de uno a varios en DynamoDB
- Principios básicos del modelado de datos en DynamoDB
- Componentes del modelado de datos en DynamoDB
- Paquetes de diseño de esquemas de modelado de datos en DynamoDB
- Prácticas recomendadas para modelar datos relacionales en DynamoDB

Colecciones de elementos: cómo modelar relaciones de uno a varios en DynamoDB

En DynamoDB, una colección de elementos es un grupo de elementos que comparten el mismo valor de clave de partición, lo que significa que los elementos están relacionados. Las colecciones de

elementos son el mecanismo principal para modelar las relaciones de uno a varios en DynamoDB. Las colecciones de elementos solo pueden existir en tablas o índices configurados para utilizar una clave primaria compuesta.

> **Note**
>
> Las colecciones de elementos pueden existir en una tabla base o en un índice secundario. Para obtener más información específicamente sobre cómo interactúan las colecciones de elementos con los índices, consulte Colecciones de elementos en los índices secundarios locales.

Considere la siguiente tabla que muestra tres usuarios diferentes y sus inventarios en el juego:

Primary key		Attributes		
Partition key: PK	**Sort key: SK**			
account1234	inventory::armor	data		
		{"armor": [{"name" : "Pauldron of the Paladin", "type" : "chest", "gear score" : 545 }, { "name" : "Greaves of the Ranger", "type" : "sword", "gear score" : 382}]}		
	inventory::weapons	data		
		{"weapons" : [{ "name" : "Sword of the Ancients", "type" : "sword", "gear score" : 320}]}		
	login-data	pw	state	last-login
		d1e8a70b5ccab1dc2f56bbf7e99f064a660c08e361a35751b9c483c88943d082	Active	1649276737
account1387	info	data		
		{"email" : "bot123@gmail.com"}		
	inventory::armor	data		
		{"armor": [{"name" : "Pauldron of the Paladin", "type" : "chest", "gear score" : 545 }, { "name" : "Greaves of the Ranger", "type" : "sword", "gear score" : 382}]}		
	login-data	pw	state	last-login
		k2g8jk0m5ppab1dc2f56bbf7e99f064a660c08e361a35751b9c464r23943i082	Banned	1649456737
account1138	info	data		
		{"email" : "luh-3417@gmail.com" }		
	login-data	pw	state	last-login
		88A41A9A62B11CCC8C120861928765A3EA41DEB9EAFE261D90F619473B89A2D4	Active	14275516966

Para algunos elementos de cada colección, la clave de clasificación es una concatenación formada por información utilizada para agrupar datos, como `inventory::armor`, `inventory::weapon` o `info`. Cada colección de elementos puede tener una combinación diferente de estos atributos como clave de clasificación. El usuario `account1234` tiene un elemento `inventory::weapons`, mientras que el usuario `account1387` no lo tiene (porque aún no ha encontrado ninguno). El usuario `account1138` solo utiliza dos elementos para su clave de clasificación (ya que aún no tiene inventario) mientras que los demás usuarios utilizan tres.

DynamoDB le permite recuperar de forma selectiva elementos de estas colecciones de elementos para hacer lo siguiente:

- Recuperar todos los elementos de un usuario concreto
- Recuperar solo un elemento de un usuario concreto
- Recuperar todos los elementos de un tipo específico pertenecientes a un usuario concreto

Acelerar las consultas mediante la organización de los datos con colecciones de elementos

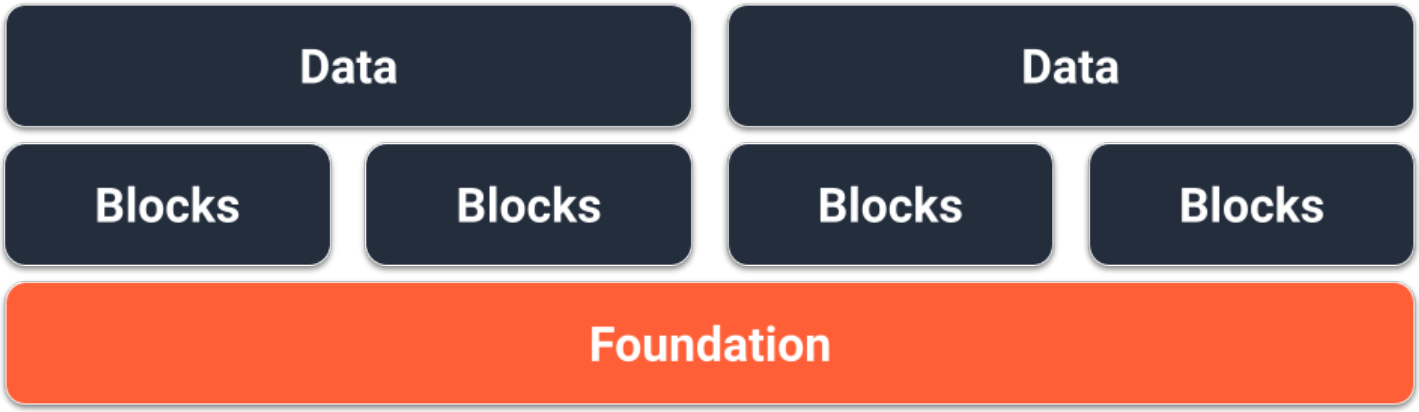
En este ejemplo, cada uno de los elementos de estas tres colecciones de elementos representa a un jugador y el modelo de datos que hemos elegido, basado en los patrones de acceso del juego y del jugador. ¿Qué datos necesita el juego? ¿Cuándo los necesita? ¿Con qué frecuencia los necesita? ¿Cuál es el costo de hacerlo de esta manera? Estas decisiones de modelado de datos se tomaron a partir de las respuestas a estas preguntas.

En este juego, se presenta al jugador una página diferente para su inventario para las armas y otra página para la armadura. Cuando el jugador abre su inventario, las armas se muestran primero porque deseamos que esa página se cargue muy rápida, mientras que las siguientes páginas del inventario pueden cargarse después. Como cada uno de estos tipos de elementos puede ser bastante grande a medida que el jugador adquiere más elementos en el juego, decidimos que cada página del inventario sería su propio elemento en la colección de elementos del jugador en la base de datos.

En la siguiente sección se describe con más detalle cómo puede interactuar con las colecciones de elementos a través de la operación `Query`.

Principios básicos del modelado de datos en DynamoDB

En esta sección se trata la capa fundamental mediante el examen de los dos tipos de diseño de tablas: tabla única y tabla múltiple.



Principios básicos de diseño de tabla única

Una opción para el principio básico de nuestro esquema de DynamoDB es el diseño de tabla única. El diseño de tabla única es un patrón que permite almacenar varios tipos (entidades) de datos en una sola tabla de DynamoDB. El objetivo es optimizar los patrones de acceso a los datos, mejorar el rendimiento y reducir los costos al eliminar la necesidad de mantener tablas múltiples y relaciones complejas entre ellas. Esto es posible porque DynamoDB almacena elementos con la misma clave de partición (lo que se conoce como recopilación de elementos) en las mismas particiones entre sí. En este diseño, los diferentes tipos de datos se almacenan como elementos en la misma tabla y cada elemento se identifica mediante una clave de clasificación única.

Primary key		Attributes	
Partition key: PK	**Sort key: SK**		
UserID	Address#USA#CA#LA#90029	data	GSI-SK
		"Street Address"	Default
	Cart#ACTIVE#Coffee	data	GSI-SK
		CoffeeSKU	2019-11-27T103324
	Cart#ACTIVE#Spice	data	GSI-SK
		SpiceSKU	2019-11-28T091245
	Cart#SAVED#Cocoa	data	GSI-SK
		CocoaSKU	2019-11-28T125642
	OrderHistory#OrderUID	data	GSI-SK
		{Order:DataMap}	2019-10-08T132612
	ProfileName	data	
		"Paul Atreides"	
	Store#StoreUID	data	GSI-SK
		Los Angeles	Active

Ventajas

- Localización de datos para admitir consultas para varios tipos de entidades en una sola llamada a la base de datos
- Reduce los costos financieros y de latencia generales de las lecturas:
  - Una sola consulta para dos elementos con un total de menos de 4 KB es 0,5 RCU de coherencia eventual
  - Dos consultas para dos elementos con un total de menos de 4 KB es 1 RCU de coherencia eventual (0,5 cada RCU)
  - El tiempo para devolver dos llamadas independientes a la base de datos será, de media, superior al de una sola llamada
- Reduce el número de tablas que hay que administrar:
  - No es necesario mantener los permisos en varios roles o políticas de IAM
  - La administración de la capacidad de la tabla se calcula de media en todas las entidades, lo que suele dar como resultado un patrón de consumo más predecible

  - El monitoreo requiere menos alarmas
  - Las claves de cifrado administradas por el cliente solo se deben rotar en una tabla
- Suaviza el tráfico hacia la tabla:
  - Al agregar varios patrones de uso a la misma tabla, el uso general tiende a ser más fluido (de la misma manera que el rendimiento de un índice bursátil tiende a ser más fluido que el de cualquier acción individual), lo que funciona mejor para lograr una mayor utilización con tablas de modos aprovisionadas

Desventajas

- La curva de aprendizaje puede ser pronunciada debido a un diseño paradójico en comparación con las bases de datos relacionales
- Los requisitos de datos deben ser coherentes en todos los tipos de entidades
  - Las copias de seguridad son todo o nada, por lo que si algunos datos no son de vital importancia, considere guardarlos en una tabla distinta
  - El cifrado de tablas se comparte entre todos los elementos. Para las aplicaciones de varios inquilinos con requisitos de cifrado de inquilinos individuales, se requeriría el cifrado del cliente
  - Las tablas con una combinación de datos históricos y datos operativos no obtendrán tantos beneficios al habilitar la clase de almacenamiento de acceso poco frecuente. Para obtener más información, consulte Clases de tablas de DynamoDB
- Todos los datos modificados se propagarán a DynamoDB Streams aunque solo se tenga que procesar un subconjunto de entidades.
  - Gracias a los filtros de eventos de Lambda, esto no afectará a la factura cuando utilice Lambda, pero supondrá un costo adicional si utiliza la Kinesis Consumer Library
- Al usar GraphQL, el diseño de una sola tabla será más difícil de implementar
- Cuando se utilizan clientes de SDK de nivel superior, como `DynamoDBMapper` de Java o Cliente mejorado, puede resultar más difícil procesar los resultados porque los elementos de la misma respuesta pueden estar asociados a clases diferentes

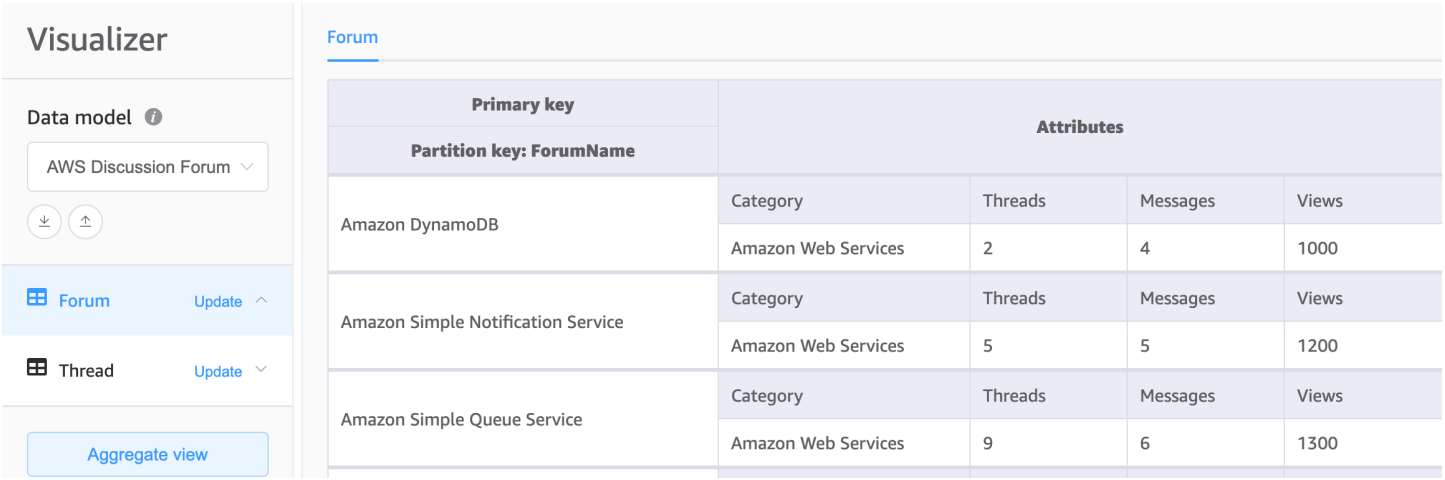
Cuándo se debe usar

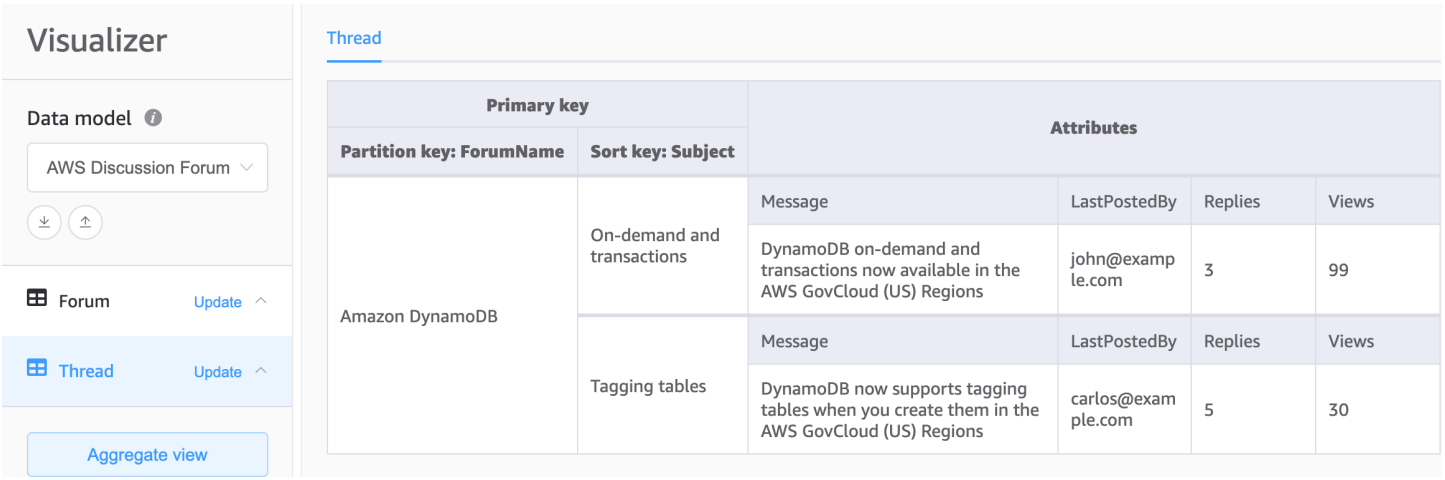
El diseño de tabla única funciona bien para aplicaciones que consultan con frecuencia varios tipos de entidades juntos o que necesitan mantener relaciones entre diferentes tipos de datos. Es

especialmente eficaz cuando los patrones de acceso se benefician de la localización de los datos y cuando se desea minimizar la sobrecarga de administración de múltiples tablas.

Principios básicos de diseño de tabla múltiple

La segunda opción para el principio básico de nuestro esquema de DynamoDB es el diseño de tabla múltiple. El diseño de tabla múltiple es un patrón que se parece más a un diseño de base de datos tradicional, en el que se almacena un único tipo (entidad) de datos en cada tabla de DynamoDB. Los datos de cada tabla seguirán organizados por clave de partición, por lo que el rendimiento dentro de un solo tipo de entidad se optimizará en función de la escalabilidad y el rendimiento, pero las consultas en varias tablas se deben realizar de forma independiente.





Ventajas

- Más fácil de diseñar para aquellos que no están acostumbrados a trabajar con un diseño de tabla única

- Implementación más sencilla de los solucionadores de GraphQL debido a que cada resolución se asigna a una sola entidad (tabla)
- Permite requisitos de datos únicos en diferentes tipos de entidades:
  - Se pueden realizar copias de seguridad de las tablas individuales que son críticas
  - El cifrado de tablas se puede administrar para cada tabla. Para las aplicaciones de múltiples inquilinos con requisitos de cifrado de inquilinos individuales, las tablas de inquilinos independientes permiten que cada cliente tenga su propia clave de cifrado
  - La clase de almacenamiento de acceso poco frecuente se puede habilitar solo en las tablas con datos históricos para obtener todos los beneficios de ahorro de costos. Para obtener más información, consulte Clases de tablas de DynamoDB
- Cada tabla tendrá su propio flujo de datos de cambios, lo que permitirá diseñar una función de Lambda dedicada para cada tipo de elemento, en lugar de un único procesador monolítico

Desventajas

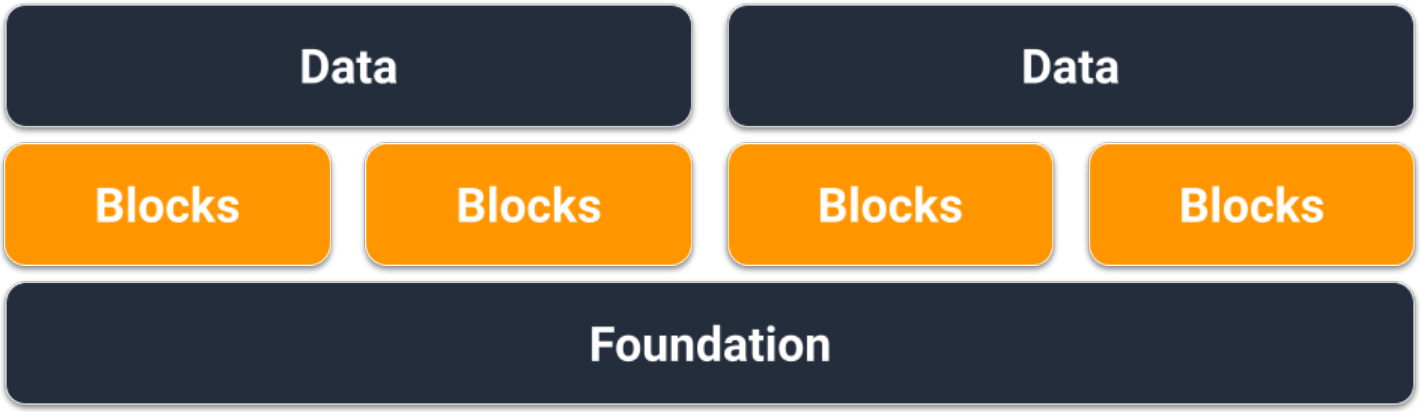
- Para los patrones de acceso que requieren datos en varias tablas, se requerirán varias lecturas de DynamoDB y es posible que sea necesario procesar o unir los datos en el código del cliente.
- Las operaciones y el monitoreo de varias tablas requieren más alarmas de CloudWatch y cada tabla se debe escalar de forma independiente
- Los permisos de cada tabla se deberán administrar de forma independiente. La adición de tablas en el futuro requerirá un cambio en los roles de IAM o políticas necesarios

Cuándo se debe usar

Si los patrones de acceso de su aplicación no tienen la necesidad de consultar varias entidades o tablas a la vez, entonces el diseño de tablas múltiples es un enfoque bueno y suficiente.

Componentes del modelado de datos en DynamoDB

En esta sección se trata la capa de componentes para ofrecerle patrones de diseño que pueda utilizar en su aplicación.



Temas

- Componente de clave de clasificación compuesta
- Complemento de tenencia múltiple
- Componente de índices dispersos
- Componente de tiempo de vida
- Componente de tiempo de vida para fines de archivado
- Componente de particiones verticales
- Escribir un componente de partición

Componente de clave de clasificación compuesta

Cuando las personas piensan en NoSQL, también pueden pensar que no es relacional. En última instancia, no hay ningún motivo para que las relaciones no puedan incluirse en un esquema de DynamoDB, simplemente tienen un aspecto diferente al de las bases de datos relacionales y las claves externas. Uno de los patrones más importantes que podemos utilizar para desarrollar una jerarquía lógica de nuestros datos en DynamoDB es una clave de clasificación compuesta. El estilo más común para diseñar una es separar cada capa de la jerarquía (capa principal > capa secundaria > capa secundaria de la secundaria) mediante un hashtag. Por ejemplo, `PARENT#CHILD#GRANDCHILD#ETC`.

Primary key	
Partition key: PK	**Sort key: SK**
UserID	CART#ACTIVE#Apples
	CART#ACTIVE#Bananas
	CART#SAVED#Oranges
	CART#SAVED#Pears
	WISH#VEGGIES#Carrots

Si bien una clave de partición en DynamoDB siempre requiere el valor exacto para consultar los datos, podemos aplicar una condición parcial a la clave de clasificación de izquierda a derecha, similar a recorrer un árbol binario.

En el ejemplo anterior, tenemos una tienda de e-commerce con un carro de la compra que debe mantenerse en todas las sesiones de usuario. Siempre que el usuario inicie sesión, es posible que desee ver todo el carro de la compra, incluidos los elementos guardados para más adelante. Sin embargo, cuando ingrese al proceso de compra, solo se deben cargar los elementos del carro activos para la compra. Dado que ambas `KeyConditions` solicitan explícitamente las claves de clasificación CART, DynamoDB simplemente ignora los datos adicionales de la lista de deseos durante la lectura. Aunque los elementos guardados y los activos forman parte del mismo carro, debemos tratarlos de forma diferente en las diferentes partes de la aplicación, por lo que aplicar una `KeyCondition` al prefijo de la clave de clasificación es la forma más optimizada de recuperar solo los datos necesarios para cada parte de la aplicación.

Características claves de este componente

- Los elementos relacionados se almacenan localmente entre sí para un acceso efectivo a los datos
- Mediante expresiones `KeyCondition`, los subconjuntos de la jerarquía se pueden recuperar de forma selectiva, lo que significa que no hay RCU desperdiciadas
- Las diferentes partes de la aplicación pueden almacenar los elementos con un prefijo específico para evitar que se sobrescriban o se produzcan conflictos de escritura

Complemento de tenencia múltiple

Muchos clientes utilizan DynamoDB para alojar datos para las aplicaciones de tenencia múltiple. Para estos casos, lo mejor es diseñar el esquema de manera que mantenga todos los datos de un único inquilino en su propia partición lógica de la tabla. Esto aprovecha el concepto de recopilación de elementos, que es un término para todos los elementos de una tabla de DynamoDB con la misma clave de partición. Para obtener más información sobre cómo DynamoDB aborda la multitenencia, consulte Multitenencia en DynamoDB.

Primary key		Attributes
Partition key: PK	**Sort key: SK**	
UserOne	PhotoID1	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
	PhotoID2	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
UserTwo	PhotoID3	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
	PhotoID4	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]
UserThree	PhotoID5	ImageURL
		https://s3.amazonaws.com/[BUCKET-NAME]/[FILE-NAME].[FILE-TYPE]

Para este ejemplo, gestionamos un sitio de alojamiento de fotos con miles de usuarios potenciales. Inicialmente, cada usuario solo cargará fotos en su propio perfil, pero de forma predeterminada no permitiremos que ningún usuario vea las fotos de ningún otro usuario. Lo ideal sería agregar un nivel adicional de aislamiento a la autorización de la llamada de cada usuario a la API para garantizar que solo soliciten datos de su propia partición, pero a nivel de esquema, son adecuadas las claves de partición únicas.

Características clave de este componente

- La cantidad de datos leídos por cualquier usuario o inquilino solo puede ser igual a la cantidad total de elementos en la partición
- La eliminación de los datos de un inquilino debido al cierre de una cuenta o a una solicitud de cumplimiento se puede hacer con tacto y de forma económica. Simplemente ejecute una

consulta en la que la clave de partición sea igual al ID de inquilino y, a continuación, ejecute una operación `DeleteItem` para cada clave principal devuelta

> **Note**
>
> Diseñado pensando en la tenencia múltiple, puede utilizar diferentes proveedores de claves de cifrado en una sola tabla para aislar los datos de forma segura. AWS El SDK de cifrado de bases de datos para Amazon DynamoDB le permite incluir el cifrado del cliente en las cargas de trabajo de DynamoDB. Puede realizar un cifrado de nivel de atributo, lo que le permite cifrar valores de atributos específicos antes de almacenarlos en la tabla de DynamoDB y buscar atributos cifrados sin descifrar previamente toda la base de datos.

Componente de índices dispersos

A veces, un patrón de acceso requiere buscar objetos que coincidan con un elemento raro o un objeto que reciba un estado (lo que requiere una respuesta escalada). En lugar de consultar estos elementos con regularidad en todo el conjunto de datos, podemos aprovechar el hecho de que los índices secundarios globales (GSI) están escasamente cargados de datos. Esto significa que solo los elementos de la tabla base que tengan los atributos definidos en el índice se replicarán en el índice.

Primary key		Attributes		
Partition key: DeviceID	**Sort key: State#Date**			
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	
		Liz	2020-04-24	
	WARNING1#2020-04-24T14:45:00	Operator	Date	
		Liz	2020-04-24	
	WARNING1#2020-04-24T14:50:00	Operator	Date	
		Liz	2020-04-24	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	
		Liz	2020-04-11	
	NORMAL#2020-04-11T09:30:00	Operator	Date	
		Sue	2020-04-11	
	WARNING2#2020-04-11T09:25:00	Operator	Date	
		Sue	2020-04-11	
	WARNING3#2020-04-11T05:55:00	Operator	Date	
		Liz	2020-04-11	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	
		Sue	2020-04-27	
	WARNING4#2020-04-27T16:15:00	Operator	Date	EscalatedTo
		Sue	2020-04-27	Sara

Primary key		Attributes	
Partition key: EscalatedTo	**Sort key: State#Date**		
Sara	WARNING4#2020-04-27T16:15:00	DeviceID	Operator
		d#11223	Sue

En este ejemplo, vemos un caso de uso de IOT en el que cada dispositivo del campo informa de un estado de forma regular. En la mayoría de los informes, esperamos que el dispositivo informe que todo está bien, pero en ocasiones puede haber un error y hay que remitirlo a un técnico de reparación. En el caso de los informes con una escalación, el atributo `EscalatedTo` se agrega al elemento, pero de lo contrario no está presente. El GSI de este ejemplo está particionado por `EscalatedTo` y, dado que el GSI trae las claves de la tabla base, aún podemos ver qué DeviceID informó del error y en qué momento.

Aunque las lecturas son más baratas que las escrituras en DynamoDB, los índices dispersos son una herramienta muy eficaz para casos de uso en los que las instancias de un tipo específico de elemento son poco frecuentes, pero son habituales las lecturas para encontrarlos.

Características claves de este componente

- Los costos de escritura y almacenamiento del GSI disperso solo se aplican a los elementos que coinciden con el patrón de claves, por lo que el costo del GSI puede ser sustancialmente menor que el de otros GSI en los que se replican todos los elementos
- Todavía se puede utilizar una clave de clasificación compuesta para reducir aún más los elementos que coinciden con la consulta deseada; por ejemplo, se puede usar una marca temporal para que la clave de clasificación solo vea los errores notificados en los últimos X minutos (`SK > 5 minutes ago, ScanIndexForward: False`)

Componente de tiempo de vida

La mayoría de los datos tienen un periodo de tiempo durante el cual se puede considerar que vale la pena guardarlos en un almacén de datos principal. Para facilitar la salida de datos de DynamoDB, cuenta con una característica denominada Tiempo de vida (TTL). La característica TTL permite definir un atributo específico en el nivel de tabla que necesita monitoreo para los elementos con una marca temporal Epoch (que está en el pasado). Esto le permite eliminar los registros caducados de la tabla de forma gratuita.

> **Note**
>
> Si usas la versión 2019.11.21 (actual) de las tablas globales y también la característica Tiempo de vida, DynamoDB replicará las eliminaciones de TTL en todas las tablas de réplica. La eliminación de TTL inicial no consume capacidad de escritura en la región donde se produce el vencimiento de TTL. Sin embargo, la eliminación de TTL replicada en las tablas

de réplica consume capacidad de escritura replicada en cada una de las regiones de réplica. En estos casos, se aplicarán los cargos pertinentes.

Primary key		Attributes	
Partition key: PK	**Sort key: MessageTimestamp**		
UserID	2030-06-30T12:12:12	TTL	Message
		1909570332	Hello
	2030-06-30T12:17:22	TTL	Message
		1909570647	DynamoDB
	2030-06-30T12:22:27	TTL	Message
		1909570947	TTL

En este ejemplo, tenemos una aplicación diseñada para permitir a un usuario crear mensajes de corta duración. Cuando se crea un mensaje en DynamoDB, el código de la aplicación establece el atributo TTL como una fecha dentro de siete días. En aproximadamente siete días, DynamoDB comprobará que la marca temporal Epoch de estos elementos pertenece al pasado y los eliminará.

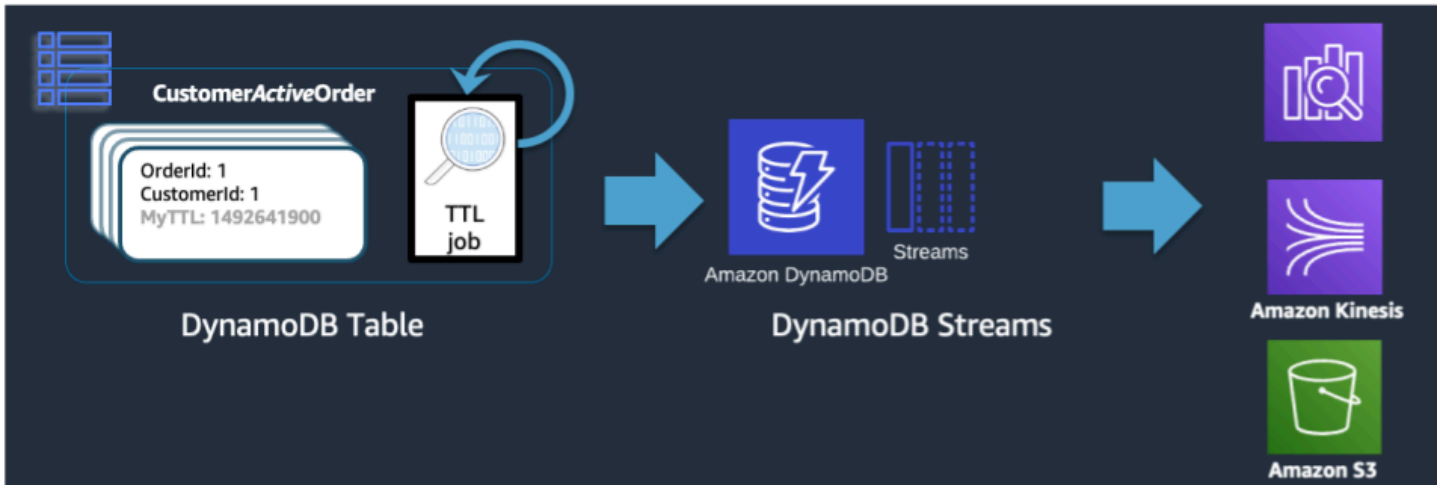
Dado que las eliminaciones realizadas por TTL son gratuitas, se recomienda encarecidamente utilizar esta característica para eliminar los datos históricos de la tabla. Esto reducirá la factura total de almacenamiento cada mes y probablemente reducirá los costos de lectura de los usuarios, ya que las consultas tendrán que recuperar menos datos. Aunque TTL está activado en el nivel de tabla, usted decide para qué elementos o entidades desea crear un atributo de TTL y en qué plazo establecer la marca temporal Epoch.

Características clave de este componente

- Las eliminaciones de TTL se ejecutan entre bastidores sin afectar al rendimiento de la tabla
- TTL es un proceso asíncrono que se ejecuta aproximadamente cada seis horas, pero puede tardar más de 48 horas en eliminarse un registro caducado
  - No confíe en las eliminaciones de TTL para casos de uso, como bloquear registros o administrar estados, si los datos obsoletos se deben limpiar en menos de 48 horas
- Puede asignar al atributo de TTL un nombre de atributo válido, pero el valor debe ser de tipo numérico

Componente de tiempo de vida para fines de archivado

Aunque TTL es una herramienta eficaz para eliminar datos antiguos de DynamoDB, en muchos casos de uso es necesario archivar los datos durante un periodo de tiempo mayor que el almacén de datos principal. En esta instancia, podemos aprovechar la eliminación cronometrada de registros de TTL para enviar los registros caducados a un almacén de datos a largo plazo.



Cuando DynamoDB elimina un TTL, se sigue introduciendo en DynamoDB Stream como un evento `Delete`. Cuando DynamoDB TTL es quien realiza la eliminación, hay un atributo en el registro de transmisión de `principal:dynamodb`. Al utilizar un suscriptor de Lambda a DynamoDB Stream, podemos aplicar un filtro de eventos solo para el atributo de la entidad principal de DynamoDB y saber que todos los registros que coincidan con ese filtro se enviarán a un almacén de archivos como Amazon Glacier.

Características clave de este componente

- Una vez que las lecturas de baja latencia de DynamoDB ya no sean necesarias para los elementos históricos, migrarlos a un servicio de almacenamiento más frío, como Amazon Glacier, puede reducir significativamente los costos de almacenamiento y, al mismo tiempo, cumplir con los requisitos de cumplimiento de datos del caso de uso
- Si los datos se almacenan en Amazon S3, se pueden utilizar herramientas de análisis rentables como Amazon Athena o Redshift Spectrum para realizar un análisis histórico de los datos

Componente de particiones verticales

Los usuarios familiarizados con una base de datos de modelos de documentos estarán familiarizados con la idea de almacenar todos los datos relacionados en un único documento JSON.

Aunque DynamoDB admite tipos de datos de JSON, no se permite ejecutar `KeyConditions` en JSON anidados. Dado que `KeyConditions` son las que dictan la cantidad de datos que se leen del disco y el número efectivo de RCU que consume una consulta, esto puede generar ineficiencias a escala. Para optimizar mejor las escrituras y lecturas de DynamoDB, recomendamos dividir las entidades individuales del documento en elementos individuales de DynamoDB, también denominados particiones verticales.

```
{
        ”UserProfile" : {
                "FirstName": "Paul",
                "LastName": "Atreides",
                "DateJoined":"1965-08-01"},
        ”Store" : {
                ”store_id": ”STOREUID",
                 "city": "Los Angeles",
                 "zip_code":"90029”}
        "ShoppingCart" : [
                {"Spice":
                        { "SKU": "SpiceSKU",
                          "CategoryID":"FictionalSpice",
                          "DateAddded " :"2019-06-11"}},
                {"EspressoBeans":
                        { "SKU": "CaffeineSKU",
                          "CategoryID":"FOODANDDRINK",
                          "DateAddded " :"2019-06-10"}}],
        "ShippingAddress" : {
                 "street_address": "1234 Arrakis Dr",
                 "city": "Los Angeles",
                 "zip_code":"90029",
                 "status": "default"}
        ”OrderHistory#OrderUID" : {
                ”ProductA": ”SKU_A",
                ”ProductB": ”SKU_B",
                "DateOrdered":”2018-09-28"}
}
```

Primary key		Attributes	
Partition key: PK	Sort key: SK		
UserID	Address#USA#CA#LA#90029	data	GSI-SK
		"Street Address"	Default
	Cart#ACTIVE#Coffee	data	GSI-SK
		CoffeeSKU	2019-11-27T103324
	Cart#ACTIVE#Spice	data	GSI-SK
		SpiceSKU	2019-11-28T091245
	Cart#SAVED#Cocoa	data	GSI-SK
		CocoaSKU	2019-11-28T125642
	OrderHistory#OrderUID	data	GSI-SK
		{Order:DataMap}	2019-10-08T132612
	ProfileName	data	
		"Paul Atreides"	
	Store#StoreUID	data	GSI-SK
		Los Angeles	Active

La partición vertical, como se muestra arriba, es un ejemplo clave del diseño de una sola tabla en acción, pero también se puede implementar en varias tablas si se desea. Dado que las facturas de DynamoDB se escriben en incrementos de 1 KB, lo ideal sería particionar el documento de forma que los elementos no superen 1 KB.

Características clave de este componente

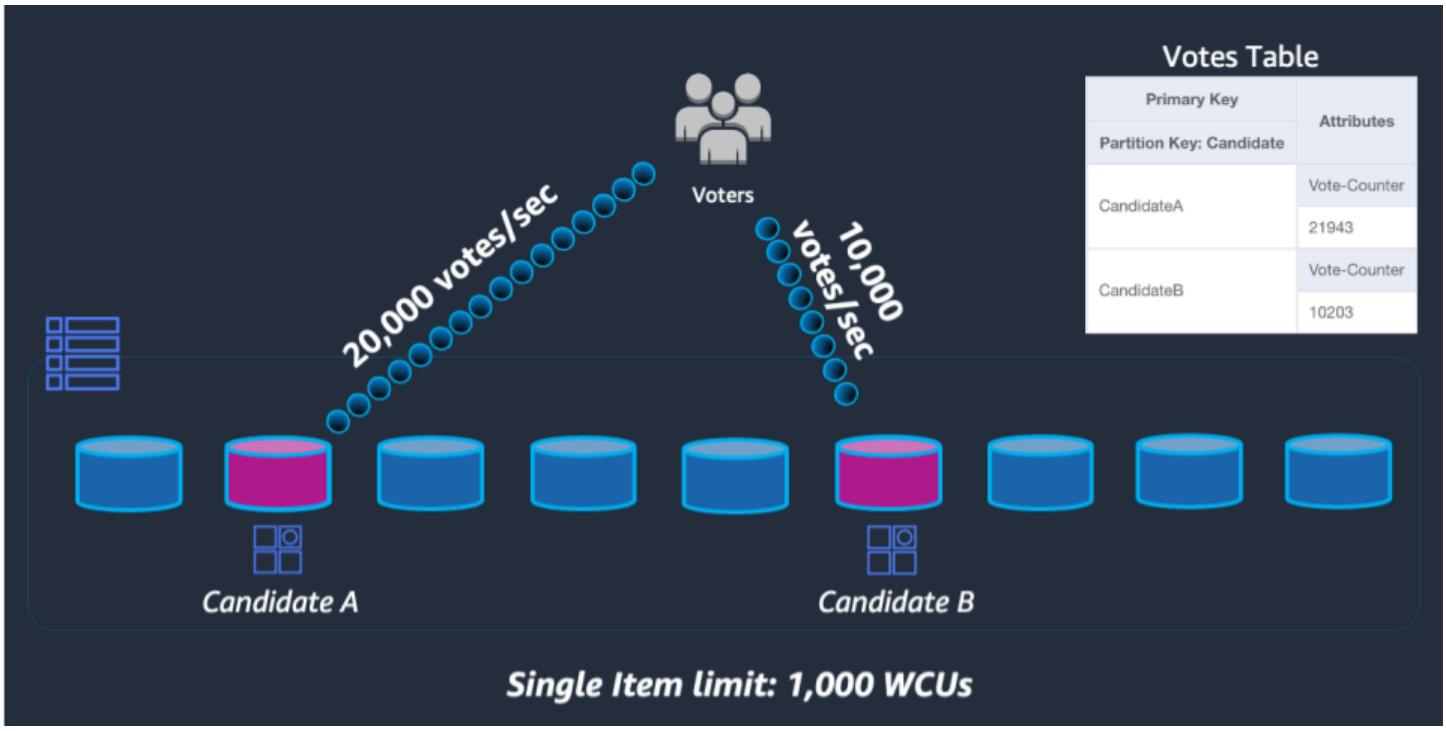
- Se mantiene una jerarquía de relaciones de datos mediante prefijos de clave de clasificación, por lo que la estructura del documento singular podría reconstruirse del lado del cliente si fuera necesario
- Los componentes singulares de la estructura de datos se pueden actualizar de forma independiente, lo que hace que las actualizaciones de elementos pequeños sean solo 1 WCU
- Al usar la clave de clasificación `BeginsWith`, la aplicación puede recuperar datos similares en una sola consulta, agregando los costos de lectura para reducir el costo total o la latencia
- Los documentos grandes pueden superar fácilmente el límite de tamaño de elemento individual de 400 KB en DynamoDB y la partición vertical ayuda a evitar este límite

Escribir un componente de partición

Uno de los pocos límites estrictos que tiene DynamoDB es la restricción del rendimiento que puede mantener una sola partición física por segundo (no necesariamente una clave de partición única). Estos límites son actualmente:

- 1000 WCU (o 1000 elementos de <= 1 KB escritos por segundo) y 3000 RCU (o 3000 lecturas de <= 4 KB por segundo) de coherencia alta o
- 6000 lecturas de <= 4 KB por segundo de coherencia final

En caso de que las solicitudes de la tabla superen alguno de estos límites, se devuelve un error al SDK del cliente de `ThroughputExceededException`, más comúnmente denominado limitación. Los casos de uso que requieran operaciones de lectura más allá de ese límite se solucionarán mejor si se coloca una memoria caché de lectura delante de DynamoDB, pero las operaciones de escritura requieren un diseño en el nivel de esquema conocido como fragmentación de escritura.



Primary Key	Attributes	
Partition Key: Candidate		
CandidateA#1	Vote-Counter	Last-Update
	10238	2019-09-30T11:35:53
CandidateA#2	Vote-Counter	Last-Update
	8452	2019-09-30T11:35:53
CandidateA#3	Vote-Counter	Last-Update
	9148	2019-09-30T11:35:53
CandidateA#4	Vote-Counter	Last-Update
	11092	2019-09-30T11:35:53

Para resolver este problema, agregaremos un número entero aleatorio al final de la clave de partición para cada participante en el código `UpdateItem` de la aplicación. El rango del generador de enteros aleatorios deberá tener un límite superior que coincida o supere la cantidad esperada de escrituras por segundo para un participante determinado dividida entre 1000. Para respaldar 20 000 votos por segundo, se vería como rand (0,19). Ahora que los datos se almacenan en particiones lógicas independientes, deben volver a combinarse en el momento de la lectura. Como los totales de votos no tienen por qué estar en tiempo real, una función de Lambda programada para leer todas las particiones de votos cada X minutos podría realizar una agregación ocasional para cada participante y volver a escribirla en un único registro total de votos para su lectura en directo.

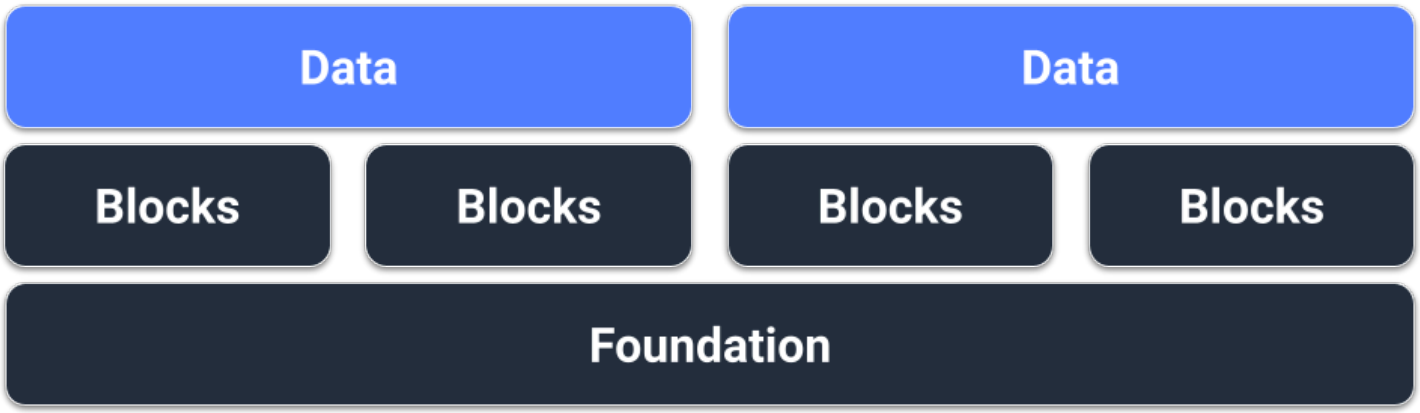
Características clave de este componente

- Para casos de uso con un rendimiento de escritura extremadamente alto para una clave de partición determinada que no se pueda evitar, las operaciones de escritura se pueden distribuir artificialmente en varias particiones de DynamoDB
- Los GSI con una clave de partición de baja cardinalidad también deberían utilizar este patrón, ya que la limitación de un GSI aplicará una contrapresión a las operaciones de escritura en la tabla base

Paquetes de diseño de esquemas de modelado de datos en DynamoDB

Obtenga información sobre los paquetes de diseño de esquema de modelado de datos para DynamoDB, que incluyen casos de uso, patrones de acceso y diseños de esquema finales para

redes sociales, perfiles de juegos, administración de reclamaciones, pagos periódicos, estado de dispositivo y tiendas en línea.



Requisitos previos

Antes de intentar diseñar nuestro esquema para DynamoDB, primero debemos recopilar algunos datos de requisitos previos sobre el caso de uso que debe admitir el esquema. A diferencia de las bases de datos relacionales, DynamoDB se fragmenta de forma predeterminada, lo que significa que los datos se alojarán en varios servidores entre bastidores, por lo que es importante diseñar teniendo en cuenta la ubicación de los datos. Tendremos que agrupar la siguiente lista para cada diseño de esquema:

- Lista de entidades (diagrama de ER)
- Volúmenes y rendimiento estimados para cada entidad
- Patrones de acceso que se deben admitir (consultas y escrituras)
- Requisitos de retención de datos

Temas

- Diseño de esquemas de redes sociales en DynamoDB
- Diseño de esquemas de perfiles de juego en DynamoDB
- Diseño del esquema del sistema de administración de reclamaciones en DynamoDB
- Diseño del esquema de pagos periódicos en DynamoDB
- Monitoreo de las actualizaciones de estado de los dispositivos en DynamoDB
- Uso de DynamoDB como almacén de datos para una tienda en línea

Diseño de esquemas de redes sociales en DynamoDB

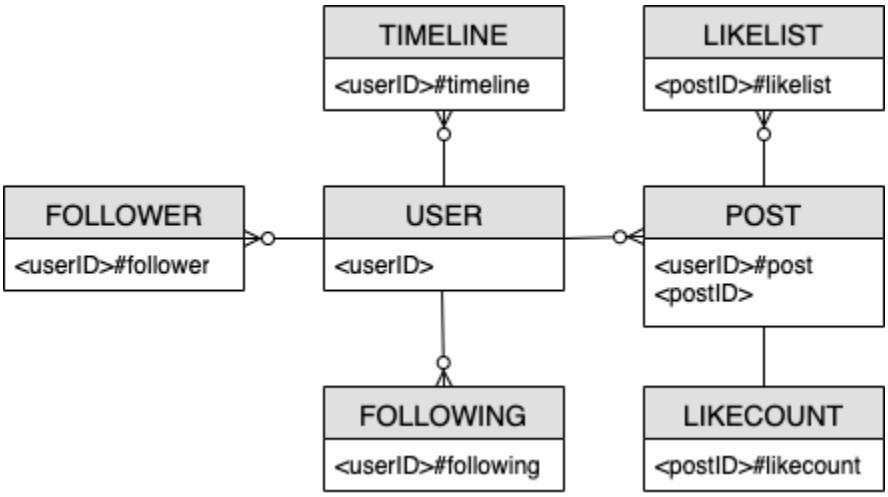
Caso de uso empresarial de redes sociales

En este caso de uso se habla del uso de DynamoDB como red social. Una red social es un servicio en línea que permite a diferentes usuarios interactuar entre sí. La red social que diseñaremos permitirá al usuario ver una cronología compuesta por las publicaciones, los seguidores, a quién sigue y las publicaciones escritas por las personas a las que sigue. Los patrones de acceso para este diseño de esquema son:

- Obtener información de usuario para un ID de usuario determinado
- Obtener la lista de seguidores de un ID de usuario determinado
- Obtener la siguiente lista de un ID de usuario determinado
- Obtener la lista de publicaciones de un ID de usuario determinado
- Obtener una lista de usuarios a los que les gusta la publicación de un ID de publicación determinada
- Obtener el recuento de “me gusta” para un ID de publicación determinado
- Obtener la cronología de un ID de usuario determinado

Diagrama de relaciones entre entidades de redes sociales

Este es el diagrama de relaciones entre entidades (ERD) que utilizaremos para el diseño del esquema de redes sociales.



Patrones de acceso de redes sociales

Estos son los patrones de acceso que consideraremos para el diseño del esquema de redes sociales.

- `getUserInfoByUserID`
- `getFollowerListByUserID`
- `getFollowingListByUserID`
- `getPostListByUserID`
- `getUserLikesByPostID`
- `getLikeCountByPostID`
- `getTimelineByUserID`

Evolución del diseño de esquemas de redes sociales

DynamoDB es una base de datos NoSQL, por lo que no permite realizar una unión, una operación que combina datos de varias bases de datos. Es posible que los clientes que no estén familiarizados con DynamoDB apliquen filosofías de diseño del sistema de administración de bases de datos relacionales (RDBMS) (como crear una tabla para cada entidad) a DynamoDB cuando no lo necesiten. El propósito del diseño de tabla única de DynamoDB es escribir los datos en un formulario previamente unido de acuerdo con el patrón de acceso de la aplicación y, a continuación, utilizarlos inmediatamente sin cálculos adicionales. Para obtener más información, consulte Diseño de tabla única frente a múltiples tablas en DynamoDB.

Ahora, vamos a ver cómo evolucionaremos nuestro diseño de esquema para abordar todos los patrones de acceso.

Paso 1: Abordar el patrón de acceso 1 (**`getUserInfoByUserID`**)

Para obtener la información de un usuario determinado, necesitaremos realizar una operación `Query` en la tabla base con una condición clave de `PK=<userID>`. La operación de consulta le permite paginar los resultados, lo que puede ser útil cuando un usuario tiene muchos seguidores. Para obtener más información acerca de Query, consulte Consulta de tablas en DynamoDB.

En nuestro ejemplo, realizamos un seguimiento de dos tipos de datos para nuestro usuario: el “recuento” y la “información”. El “recuento” de un usuario refleja cuántos seguidores tiene, cuántos

usuarios sigue y cuántas publicaciones ha creado. La “información” de un usuario refleja su información personal, como su nombre.

Vemos estos dos tipos de datos representados por los dos elementos siguientes. El elemento que tiene “recuento” en su clave de clasificación (SK) tiene más probabilidades de cambiar que el elemento con “información”. DynamoDB considera el tamaño del elemento tal como aparece antes y después de la actualización y el rendimiento aprovisionado consumido reflejará el mayor de estos tamaños de elemento. Aunque se actualice tan solo un subconjunto de atributos del elemento, `UpdateItem` consumirá la cantidad total de rendimiento provisionado (el mayor de los tamaños de elemento de "antes" y "después"). Puede obtener los elementos mediante una sola operación `Query` y usar `UpdateItem` para agregar o restar atributos numéricos existentes.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...

Paso 2: Abordar el patrón de acceso 2 (**`getFollowerListByUserID`**)

Para obtener una lista de los usuarios que siguen a un usuario determinado, necesitaremos `Query` la tabla base con una condición de clave de `PK=<userID>#follower`.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				

Paso 3: Abordar el patrón de acceso 3 (**`getFollowingListByUserID`**)

Para obtener una lista de los usuarios que sigue un usuario determinado, necesitaremos `Query` la tabla base con una condición de clave de `PK=<userID>#following`. A continuación, puede utilizar una operación `TransactWriteItems` para agrupar varias solicitudes y hacer lo siguiente:

- Agregue el usuario A a la lista de seguidores del usuario B y, a continuación, incremente el número de seguidores del usuario B en uno.

- Agregue el usuario B a la lista de seguidores del usuario A y, a continuación, incremente el número de seguidores del usuario A en uno.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				

Paso 4: Abordar el patrón de acceso 4 (**`getPostListByUserID`**)

Para obtener una lista de las publicaciones creadas por un usuario determinado, necesitaremos `Query` la tabla base con una condición de clave de `PK=<userID>#post`. Una cosa importante a tener en cuenta es que los ID de publicación de un usuario deben ser incrementales: el segundo valor del ID de publicación debe ser mayor que el primer valor del ID de publicación (ya que los usuarios desean ver las publicaciones de forma ordenada). Para ello, puede generar ID de publicación en función de un valor temporal, como un identificador ordenable lexicográficamente único y universal (ULID).

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	

Paso 5: Abordar el patrón de acceso 5 (**`getUserLikesByPostID`**)

Para obtener una lista de los usuarios a los que les gusta una publicación de un usuario determinado, necesitaremos `Query` la tabla base con una condición de clave de `PK=<postID>#likelist`. Este enfoque es el mismo patrón que utilizamos para recuperar el seguidor y las listas de seguidores en el patrón de acceso 2 (`getFollowerListByUserID`) y el patrón de acceso 3 (`getFollowingListByUserID`).

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				

Paso 6: Abordar el patrón de acceso 6 (**`getLikeCountByPostID`**)

Para obtener un recuento de los “me gusta” de una publicación determinada, tendremos que realizar una operación `GetItem` en la tabla base con una condición clave de `PK=<postID>#likecount`. Este patrón de acceso puede provocar problemas de limitación cuando un usuario con muchos seguidores (como una celebridad) crea una publicación, ya que la limitación se produce cuando el rendimiento de una partición supera los 1000 WCU por segundo. Este problema no se debe a DynamoDB, solo aparece en DynamoDB, ya que se encuentra al final de la pila de software.

Debe evaluar si es realmente esencial que todos los usuarios vean el recuento de me gusta simultáneamente o si se puede producir de forma gradual con el tiempo. En general, el recuento de “me gusta” de una publicación no necesita ser inmediatamente preciso al 100 %. Puede implementar esta estrategia poniendo una cola entre la aplicación y DynamoDB para que las actualizaciones se realicen periódicamente.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				
p#12345#likecount	"count"	etc			
		100			

Paso 7: Abordar el patrón de acceso 7 (**`getTimelineByUserID`**)

Para obtener la cronología de un usuario determinado, necesitaremos realizar una operación `Query` en la tabla base con una condición de clave de `PK=<userID>#timeline`. Vamos a tener en cuenta un escenario en el que los seguidores de un usuario necesitan ver su publicación de forma sincrónica. Cada vez que un usuario escribe una publicación, se lee su lista de seguidores y el ID de usuario e ID de publicación se ingresan lentamente en la clave de la cronología de todos sus seguidores. A continuación, cuando se inicie la aplicación, podrá leer la clave de cronología con la operación `Query` y llenar la pantalla de la cronología con una combinación de ID de usuario e ID de publicación mediante la operación `BatchGetItem` para cualquier elemento nuevo. No puede leer la cronología con una llamada a la API, pero esta es una solución más rentable si las publicaciones se pueden editar con frecuencia.

La cronología es donde se muestran las publicaciones recientes, por lo que necesitaremos una forma de limpiar las antiguas. En lugar de utilizar WCU para eliminarlas, puede utilizar la característica TTL de DynamoDB para hacerlo de forma gratuita.

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				
p#12345#likecount	"count"	etc			
		100			
u#12345#timeline	p#34567#u#56789	ttl			
		1571827560			
	p#45678#u#67890	ttl			
		1571827560			
	p#56789#u#78901	ttl			
		1571827560			

En la tabla siguiente se resumen todos los patrones de acceso y cómo los aborda el diseño del esquema:

Patrón de acceso	Tabla base/ GSI/LSI	Operación	Valor de clave de partición	Valor de la clave de clasificación	Otras condiciones/ filtros
getUserInfoByUserID	Tabla de base	Consultar	PK=<userID>		

Patrón de acceso	Tabla base/ GSI/LSI	Operación	Valor de clave de partición	Valor de la clave de clasificación	Otras condiciones/ filtros
getFollowerListByUserID	Tabla de base	Consultar	PK=<userID>#follower		
getFollowingListByUserID	Tabla de base	Consultar	PK=<userID>#following		
getPostListByUserID	Tabla de base	Consultar	PK=<userID>#post		
getUserLikesByPostID	Tabla de base	Consultar	PK=<postID>#likelist		
getLikeCountByPostID	Tabla de base	GetItem	PK=<postID>#likecount		
getTimelineByUserID	Tabla de base	Consultar	PK=<userID>#timeline		

Esquema final de red social

Este es el diseño final del esquema. Para descargar este diseño de esquema como un archivo JSON, consulte los ejemplos de DynamoDB en GitHub.

Tabla base:

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
u#12345	"count"	follower#	following#	post#	
		3000000000	971	4945	
	"info"	name	content	imageUrl	etc
		hyuklee	My name is Hyuk Lee	s3://....	...
u#12345#follower	u#23456				
	u#34567				
	u#45678				
u#12345#following	u#56789				
	u#67890				
	u#78912				
u#12345#post	p#12345	content	imageUrl	timestamp	
		content for a post	s3://....	1571827560	
	p#23456	content	imageUrl	timestamp	
		content for a post	s3://....	1571827561	
p#12345#likelist	u#23456				
	u#34567				
	u#45678				
p#12345#likecount	"count"	etc			
		100			
u#12345#timeline	p#34567#u#56789	ttl			
		1571827560			
	p#45678#u#67890	ttl			
		1571827560			
	p#56789#u#78901	ttl			
		1571827560			

Uso de NoSQL Workbench con este diseño de esquema

Puede importar este esquema final en NoSQL Workbench, una herramienta visual que proporciona características de modelado de datos, visualización de datos y desarrollo de consultas para DynamoDB, a fin de explorar y editar más a fondo el nuevo proyecto. Para comenzar, siga estos pasos:

1. Descargue NoSQL Workbench. Para obtener más información, consulte the section called “Descargar”.
2. Descargue el archivo de esquema JSON que se muestra anteriormente, que ya está en el formato de modelo NoSQL Workbench.

3. Importe el archivo de esquema JSON en NoSQL Workbench. Para obtener más información, consulte the section called “Importación de un modelo existente”.
4. Una vez que haya importado en NOSQL Workbench, podrá editar el modelo de datos. Para obtener más información, consulte the section called “Edición de un modelo existente”.
5. Para visualizar el modelo de datos, agregar datos de ejemplo o importar datos de ejemplo de un archivo CSV, utilice la característica Visualizador de datos de NoSQL Workbench.

Diseño de esquemas de perfiles de juego en DynamoDB

Caso de uso empresarial del perfil de juego

En este caso de uso se habla del uso de DynamoDB para almacenar los perfiles de los jugadores de un sistema de juego. Los usuarios (en este caso, los jugadores) necesitan crear perfiles antes de poder interactuar con muchos juegos modernos, especialmente los que son en línea. Los perfiles de juego suelen incluir lo siguiente:

- Información básica, como el nombre de usuario
- Datos del juego, como elementos y equipos
- Registros de juegos, como tareas y actividades
- Información social, como listas de amigos

Para cumplir con los requisitos detallados de acceso a las consultas de datos para esta aplicación, las claves principales (clave de partición y clave de clasificación) utilizarán nombres genéricos (PK y SK) para sobrecargarlas con varios tipos de valores, como veremos a continuación.

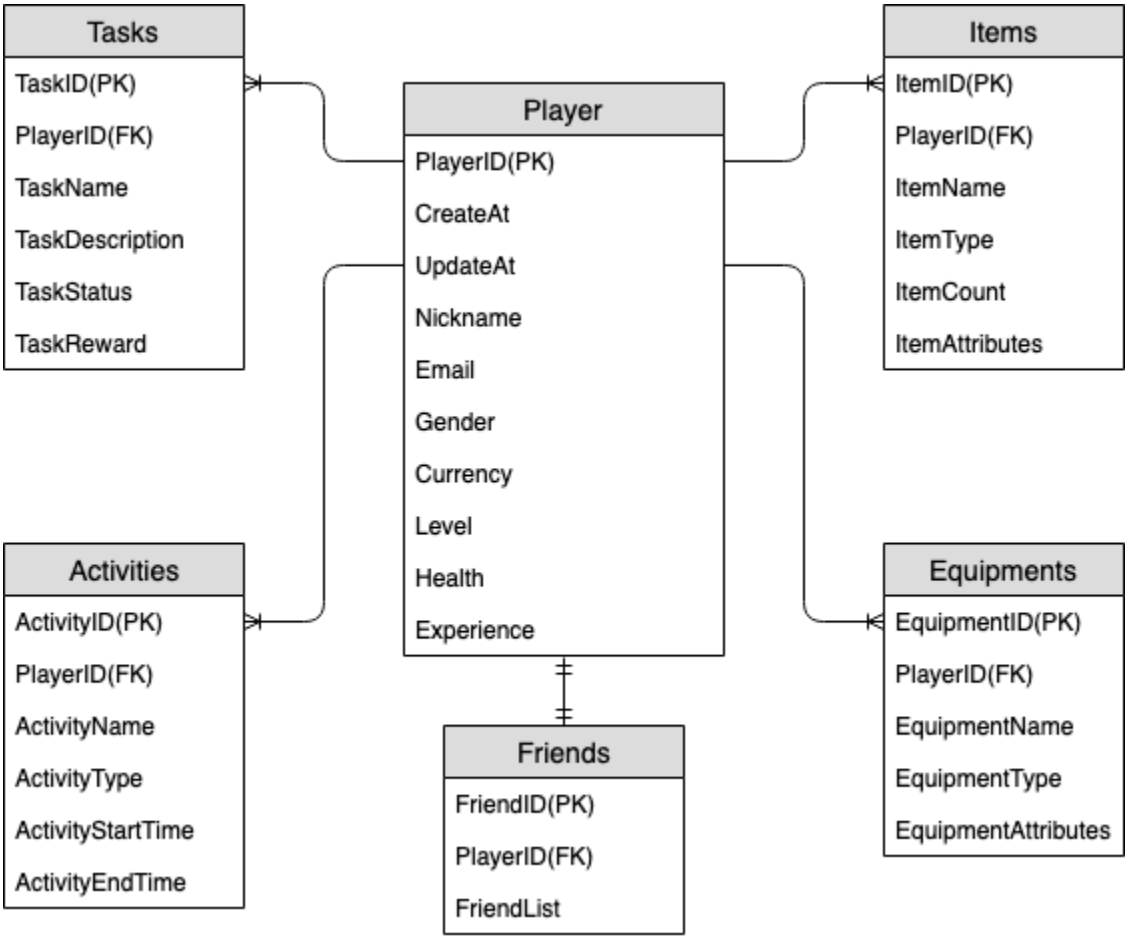
Los patrones de acceso para este diseño de esquema son:

- Obtener la lista de amigos de un usuario
- Obtener toda la información de un jugador
- Obtener la lista de elementos de un usuario
- Obtener un elemento específico de la lista de elementos del usuario
- Actualizar el personaje de un usuario
- Actualizar el recuento de elementos de un usuario

El tamaño del perfil de juego variará en los diferentes juegos. Si los valores de atributo grandes se comprimen, es posible que se ajusten a los límites de los elementos de DynamoDB y se reduzcan los costos. La estrategia de gestión del rendimiento dependería de varios factores, como el número de jugadores, el número de partidos jugados por segundo y la estacionalidad de la carga de trabajo. Normalmente, en el caso de un juego recién lanzado, se desconoce el número de jugadores y el nivel de popularidad, por lo que empezaremos con el modo de rendimiento bajo demanda.

Diagrama de relación entre entidades del perfil de juego

Este es el diagrama de relaciones entre entidades (ERD) que utilizaremos para el diseño del esquema del perfil de juego.



Patrones de acceso a los perfiles de juego

Estos son los patrones de acceso que consideraremos para el diseño del esquema de redes sociales.

- `getPlayerFriends`

- `getPlayerAllProfile`
- `getPlayerAllItems`
- `getPlayerSpecificItem`
- `updateCharacterAttributes`
- `updateItemCount`

Evolución del diseño del esquema de perfiles de juego

En el ERD anterior, podemos ver que este es un tipo de modelado de datos de relación de uno a varios. En DynamoDB, los modelos de datos de uno a varios se pueden organizar en recopilaciones de elementos, lo que es diferente de las bases de datos relacionales tradicionales, en las que se crean varias tablas y se vinculan mediante claves externas. Una recopilación de elementos es un grupo de elementos que comparten el mismo valor de clave de partición, pero tienen valores de clave de clasificación diferentes. Dentro de una recopilación de elementos, cada elemento tiene un valor de clave de clasificación único que lo distingue de otros elementos. Con esto en mente, vamos a usar el siguiente patrón para los valores `HASH` y `RANGE` para cada tipo de entidad.

Para empezar, utilizamos nombres genéricos como `PK` y `SK` para almacenar diferentes tipos de entidades en la misma tabla a fin de preparar el modelo para el futuro. Para una mejor legibilidad, podemos incluir prefijos para indicar el tipo de datos o incluir un atributo arbitrario llamado `Entity_type` o `Type`. En el ejemplo actual, utilizamos una cadena que comienza con `player` para almacenar `player_ID` como `PK`; usar `entity name#` como prefijo de `SK` y añadir un atributo `Type` para indicar de qué tipo de entidad es este dato. Esto nos permitirá almacenar más tipos de entidades en el futuro y utilizar tecnologías avanzadas como sobrecarga de GSI y GSI disperso para cumplir con más patrones de acceso.

Empecemos a implementar los patrones de acceso. Los patrones de acceso, como agregar jugadores y agregar equipo, se pueden realizar a través de la operación `PutItem`, por lo que podemos ignorarlos. En este documento, nos centraremos en los patrones de acceso típicos mostrados anteriormente.

Paso 1: Abordar el patrón de acceso 1 (**`getPlayerFriends`**)

Abordamos el patrón de acceso 1 (`getPlayerFriends`) con este paso. En nuestro diseño actual, la amistad es simple y el número de amigos en el juego es pequeño. Para simplificar, utilizamos un tipo de datos de lista para almacenar las listas de amigos (modelo 1:1). En este diseño, utilizamos `GetItem` para satisfacer este patrón de acceso. En la operación `GetItem`, proporcionamos

explícitamente la clave de partición y el valor de la clave de clasificación para obtener un elemento específico.

Sin embargo, si un juego tiene un gran número de amigos y las relaciones entre ellos son complejas (por ejemplo, las amistades son bidireccionales con un componente de invitación y de aceptación), sería necesario utilizar una relación de varios a varios para almacenar a cada amigo de forma individual, a fin de escalar hasta un tamaño ilimitado de la lista de amigos. Y si el cambio de amistad implica operar con varios elementos al mismo tiempo, las transacciones de DynamoDB se pueden usar para agrupar varias acciones y enviarlas como una sola operación de todo o nada `TransactWriteItems` o `TransactGetItems`.

Primary key		Attributes	
Partition key: PK	**Sort key: SK**		
player001	FRIENDS#player001	Type	FriendList
		Friends	[{"M": {"FriendId": {"S":"player002"},"FriendName": {"S":"Alice"}}}, {"M":{"FriendId": {"S":"player003"},"FriendName": {"S":"Bob"}}}]

Paso 2: Abordar los patrones de acceso 2 (**`getPlayerAllProfile`**), 3 (**`getPlayerAllItems`**) y 4 (**`getPlayerSpecificItem`**)

Mediante este paso, abordamos los patrones de acceso 2 (`getPlayerAllProfile`), 3 (`getPlayerAllItems`) y 4 (`getPlayerSpecificItem`). Lo que estos tres patrones de acceso tienen en común es una consulta de rango, que utiliza la operación `Query`. En función del alcance de la consulta, se utilizan Claves de condición y Expresiones de filtro, que se utilizan normalmente en el desarrollo práctico.

En la operación Query, proporcionamos un valor único para la clave de partición y obtenemos todos los elementos con ese valor de clave de partición. El patrón de acceso 2 (`getPlayerAllProfile`) se implementa de esta manera. De forma opcional, podemos agregar una expresión de condición de clave de clasificación, una cadena que determina los elementos que se van a leer de la tabla. El patrón de acceso 3 (`getPlayerAllItems`) se implementa al agregar la condición clave de clave de clasificación begins_with `ITEMS#`. Además, para simplificar el desarrollo del lado de la aplicación, podemos usar expresiones de filtro para implementar el patrón de acceso 4 (`getPlayerSpecificItem`).

Este es un ejemplo de pseudocódigo que utiliza una expresión de filtro que filtra los elementos de la categoría `Weapon`:

```
filterExpression: "ItemType = :itemType"
expressionAttributeValues: {":itemType": "Weapon"}
```

Primary key		Attributes				
Partition key: PK	**Sort key: SK**					
player001	ITEMS#001	Type	ItemName	ItemType	ItemCount	ItemAttributes
		Item	Health Potion	Consumable	5	{"M":{"HP": {"N":"50"}}}
	ITEMS#002	Type	ItemName	ItemType	ItemCount	ItemAttributes
		Item	Armor of the Knight	Armor	1	{"M":{"DEF": {"N":"100"}}}
		Type	ItemName	ItemType	ItemCount	ItemAttributes
	ITEMS#003	Item	Sword of the Dragon	Weapon	1	{"M":{"ATK": {"N":"100"},"DEF" :{"N":"50"}}}

> **Note**
>
> Una expresión de filtro se aplica después de que Query finalice, pero antes de devolver los resultados al cliente. Por consiguiente, una Query consume la misma cantidad de capacidad de lectura aunque se especifique una expresión de filtro.

Si el patrón de acceso consiste en consultar un conjunto de datos grande y filtrar una gran cantidad de datos para conservar solo un pequeño subconjunto de datos, el enfoque adecuado consiste en diseñar la clave de partición y la clave de clasificación de DynamoDB de forma más eficaz. Por ejemplo, en el ejemplo anterior para obtener un determinado `ItemType`, si hay muchos elementos para cada jugador y la consulta de un determinado `ItemType` es un patrón de acceso típico, sería más eficiente incluir `ItemType` en SK como una clave compuesta. El modelo de datos tendría este aspecto: `ITEMS#ItemType#ItemId`.

Paso 3: Abordar patrones de acceso 5 (**`updateCharacterAttributes`**) y 6 (**`updateItemCount`**)

Mediante este paso, abordamos los patrones de acceso 5 (`updateCharacterAttributes`) y 6 (`updateItemCount`). Cuando el jugador necesite modificar el personaje, por ejemplo, reducir la moneda o modificar la cantidad de un arma determinada en los elementos, use `UpdateItem` para implementar estos patrones de acceso. Para actualizar la moneda de un jugador y asegurarnos de que nunca baje de la cantidad mínima, podemos agregar una the section called “Ejemplo de la

CLI" para reducir el saldo solo si es superior o igual a la cantidad mínima. Este es un ejemplo de pseudocódigo:

```
UpdateExpression: "SET currency = currency - :amount"
ConditionExpression: "currency >= :minAmount"
```

Primary key		Attributes									
Partition key: PK	Sort key: SK	Type	CreatedAt	UpdatedAt	Nickname	Email	Gender	Avatar	Currency	PlayerLevel	PlayerHealth	PlayerExperience
player001	#METADATA #player001	Metadata	1618500000	1620000000	John	john@example.com	male	s3://gaming-blki65wn3bgc-lob-avatar/player001.png	500 Updated to 500-Amount	10	80	1000

Al desarrollar con DynamoDB y utilizar Contadores atómicos para reducir el inventario, podemos garantizar la idempotencia mediante el uso de un bloqueo positivo. Este es un ejemplo de pseudocódigo para contadores atómicos:

```
UpdateExpression: "SET ItemCount = ItemCount - :incr"
expression-attribute-values: '{":incr":{"N":"1"}}'
```

Primary key		Attributes				
Partition key: PK	Sort key: SK	Type	ItemName	ItemType	ItemCount	ItemAttributes
player001	ITEMS#001	Item	Health Potion	Consumable	5 Updated to 4	{"M":{"HP":{"N":"50"}}}

Además, en un escenario en el que el jugador compra un elemento con moneda, todo el proceso debe deducir la moneda y agregar un elemento al mismo tiempo. Podemos usar las transacciones de DynamoDB para agrupar varias acciones y enviarlas como una sola operación de todo o nada `TransactWriteItems` o `TransactGetItems`. `TransactWriteItems` es una operación de escritura sincrónica e idempotente que agrupa hasta 100 acciones de escritura en una sola operación de todo o nada. Las acciones se realizan atómicamente, de tal forma que se llevan a cabo correctamente todas o ninguna de ellas. Las transacciones ayudan a eliminar el riesgo de duplicación o desaparición de divisas. Para obtener más información acerca de las transacciones, consulte Ejemplo de transacciones en DynamoDB.

En la tabla siguiente se resumen todos los patrones de acceso y cómo los aborda el diseño del esquema:

Patrón de acceso	Tabla base/ GSI/LSI	Operación	Valor de clave de partición	Valor de la clave de clasificación	Otras condiciones/ filtros
getPlayer Friends	Tabla de base	GetItem	PK=PlayerID	SK="FRIENDS#playerID"	
getPlayer AllProfile	Tabla de base	Consultar	PK=PlayerID		
getPlayer AllItems	Tabla de base	Consultar	PK=PlayerID	SK begins_with "ITEMS#"	
getPlayer SpecificItem	Tabla de base	Consultar	PK=PlayerID	SK begins_with "ITEMS#"	filterExpression: "ItemType = :itemType " expressionAttributeValues: { ":itemType": "Weapon" }
updateCharacterAttributes	Tabla de base	UpdateItem	PK=PlayerID	SK="#METADATA#playerID"	UpdateExpression: "SET currency = currency - :amount" ConditionExpression: "currency >= :minAmount"
updateItemCount	Tabla de base	UpdateItem	PK=PlayerID	SK ="ITEMS#ItemID"	update-expression: "SET ItemCount

Patrón de acceso	Tabla base/ GSI/LSI	Operación	Valor de clave de partición	Valor de la clave de clasificación	Otras condiciones/ filtros
					= ItemCount - :incr" expression-attribute-values : '{":incr": {"N":"1"}}'

Esquema final del perfil de juego

Este es el diseño final del esquema. Para descargar este diseño de esquema como un archivo JSON, consulte los ejemplos de DynamoDB en GitHub.

Tabla base:

Primary key		Attributes										
Partition key: PK	Sort key: SK											
player001	#METADATA #player001	Type	CreatedAt	UpdatedAt	Nickname	Email	Gender	Avatar	Currency	PlayerLevel	PlayerHealth	PlayerExperience
		Metadata	1618500000	1620000000	John	john@example.com	male	s3://gaming-blki65wn3bgc-lob-avatar/player001.png	500	10	80	1000
	ACTIVITY#001	Type	ActivityEndTime	ActivityName	ActivityReward	ActivityStartTime	ActivityType					
		Activity	1647475199	Hunting Trip	{"M":{"Gold":{"N":"50"},"XP":{"N":"200"}}}	1647388800	Hunting					
	ACTIVITY#002	Type	ActivityEndTime	ActivityName	ActivityReward	ActivityStartTime	ActivityType					
		Activity	1647647999	Mining Adventure	{"M":{"Gold":{"N":"1000"},"XP":{"N":"500"}}}	1647561600	Mining					
	ACTIVITY#003	Type	ActivityEndTime	ActivityName	ActivityReward	ActivityStartTime	ActivityType					
		Activity	1647820799	Arena Challenge	{"M":{"Gold":{"N":"2000"},"XP":{"N":"1000"}}}	1647734400	Arena					
	EQUIPMENTS#001	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Sword of the Dragon	Weapon	{"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}}							
	EQUIPMENTS#001EQUIPMENTS#002	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Armor of the Knight	Armor	{"M":{"DEF":{"N":"100"}}}							
	EQUIPMENTS#003	Type	EquipmentName	EquipmentType	EquipmentAttributes							
		Equipment	Ring of the Mage	Accessory	{"M":{"SP":{"N":"50"}}}							
	FRIENDS#player001	Type	FriendList									
		Friends	[{"M":{"FriendId":{"S":"player002"},"FriendName":{"S":"Alice"}}},{"M":{"FriendId":{"S":"player003"},"FriendName":{"S":"Bob"}}}]									
	ITEMS#001	Type	ItemName	ItemType	ItemCount	ItemAttributes						
		Item	Health Potion	Consumable	5	{"M":{"HP":{"N":"50"}}}						
	ITEMS#002	Type	ItemName	ItemType	ItemCount	ItemAttributes						
		Item	Armor of the Knight	Armor	1	{"M":{"DEF":{"N":"100"}}}						
	ITEMS#003	Type	ItemName	ItemType	ItemCount	ItemAttributes						
		Item	Sword of the Dragon	Weapon	1	{"M":{"ATK":{"N":"100"},"DEF":{"N":"50"}}}						
	TASK#001	Type	TaskName	TaskDescription	TaskStatus	TaskReward						
		Task	Find the Lost TreasureFind the Lost Treasure	Get clues from a lost adventurer and find the lost treasure.	InProgress	{"M":{"Gold":{"N":"100"},"XP":{"N":"50"}}}						
	TASK#002	Type	TaskName	TaskDescription	TaskStatus	TaskReward						
		Task	Defeat Magic Monsters	Go to the Magic Forest and defeat three magic monsters.	Completed	{"M":{"Gold":{"N":"200"},"XP":{"N":"100"}}}						
	TASK#003	Type	TaskName	TaskDescription	TaskStatus	TaskReward						
		Task	Rescue the Princess	Go to the Demon King's Castle and rescue the princess who is being held captive by the Demon King.	Available	{"M":{"Gold":{"N":"500"},"XP":{"N":"200"}}}						

Uso de NoSQL Workbench con este diseño de esquema

Puede importar este esquema final en NoSQL Workbench, una herramienta visual que proporciona características de modelado de datos, visualización de datos y desarrollo de consultas para DynamoDB, a fin de explorar y editar más a fondo el nuevo proyecto. Para comenzar, siga estos pasos:

1. Descargue NoSQL Workbench. Para obtener más información, consulte the section called “Descargar”.
2. Descargue el archivo de esquema JSON que se muestra anteriormente, que ya está en el formato de modelo NoSQL Workbench.
3. Importe el archivo de esquema JSON en NoSQL Workbench. Para obtener más información, consulte the section called “Importación de un modelo existente”.
4. Una vez que haya importado en NOSQL Workbench, podrá editar el modelo de datos. Para obtener más información, consulte the section called “Edición de un modelo existente”.
5. Para visualizar el modelo de datos, agregar datos de ejemplo o importar datos de ejemplo de un archivo CSV, utilice la característica Visualizador de datos de NoSQL Workbench.

Diseño del esquema del sistema de administración de reclamaciones en DynamoDB

Caso de uso empresarial del sistema de administración de reclamaciones

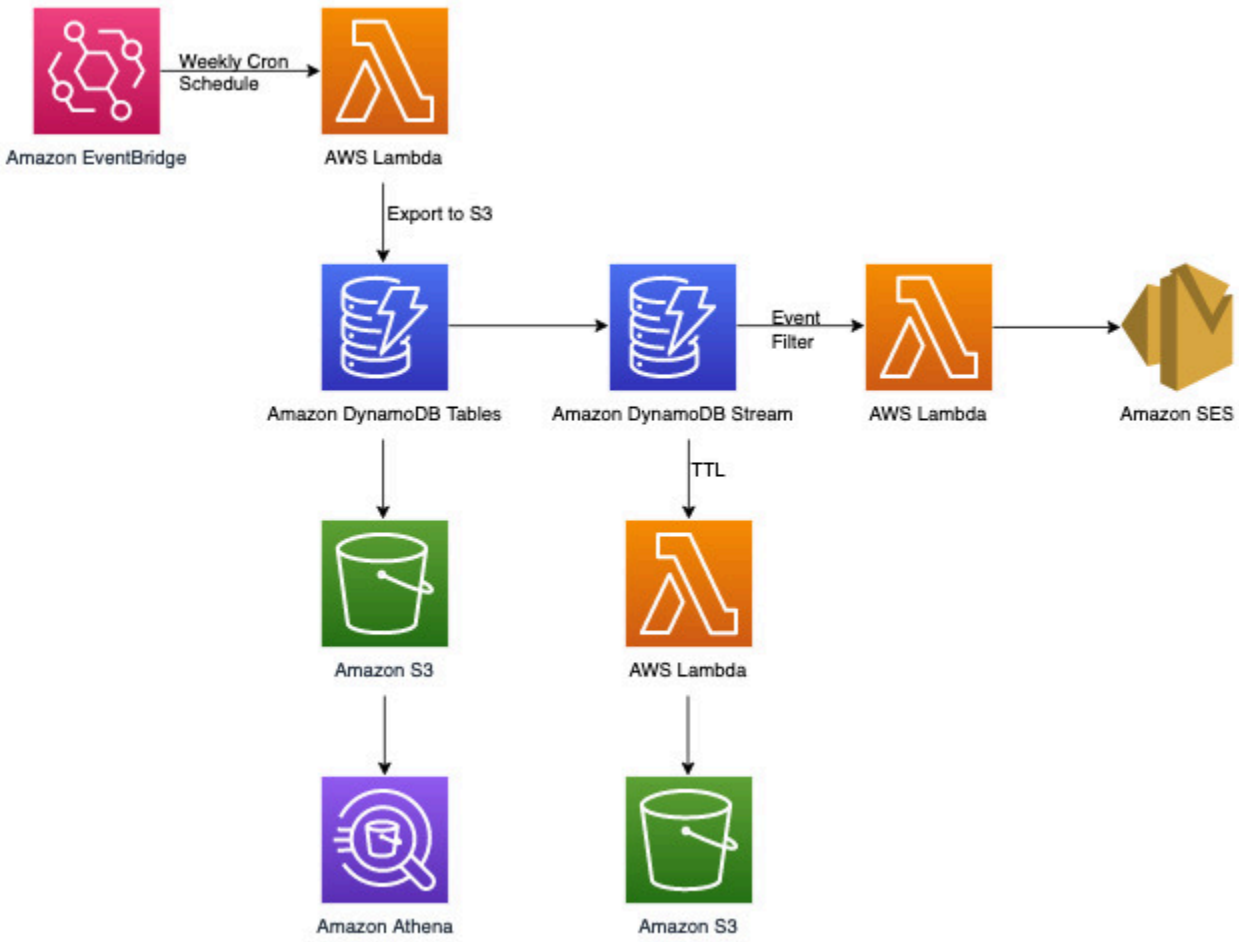
DynamoDB es una base de datos muy adecuada para el caso de uso de un sistema de administración de reclamaciones (o un centro de contacto), ya que la mayoría de los patrones de acceso asociados a ellos serían búsquedas transaccionales basadas en clave-valor. Los patrones de acceso típicos en este escenario serían:

- Crear y actualizar reclamaciones
- Escalar una reclamación
- Crear y leer comentarios sobre una reclamación
- Obtener todas las reclamaciones de un cliente
- Obtener todos los comentarios de un agente y obtener todos los escalados

Algunos comentarios pueden llevar asociada una descripción de la reclamación o de la solución. Aunque todos estos son patrones de acceso de clave-valor, puede haber requisitos adicionales como el envío de notificaciones cuando se agrega un nuevo comentario a una reclamación o la ejecución de consultas analíticas para hallar la distribución de reclamaciones por gravedad (o el rendimiento de los agentes) por semana. Un requisito adicional relacionado con la administración del ciclo de vida o el cumplimiento de la normativa sería archivar los datos de la reclamación transcurridos tres años desde su registro.

Diagrama de arquitectura del sistema de administración de reclamaciones

A continuación, se muestra el diagrama de arquitectura del sistema de administración de reclamaciones. Este diagrama muestra las diferentes integraciones de Servicio de AWS que utiliza el sistema de administración de reclamaciones.



Aparte de los patrones de acceso transaccional de clave-valor que trataremos más adelante en la sección de modelado de datos de DynamoDB, tenemos tres requisitos no transaccionales. El diagrama de arquitectura anterior puede desglosarse en los tres flujos de trabajo siguientes:

1. Enviar una notificación cuando se agregue un nuevo comentario a una reclamación
2. Ejecutar consultas analíticas sobre los datos semanales
3. Archivar datos de más de tres años

Echemos un vistazo más a fondo a cada una de ellas.

Enviar una notificación cuando se agregue un nuevo comentario a una reclamación

Podemos utilizar el siguiente flujo de trabajo para cumplir este requisito:

DynamoDB Streams es un mecanismo de captura de datos de cambios para registrar toda la actividad de escritura en sus tablas de DynamoDB. Puede configurar funciones de Lambda para desencadenar algunos o todos estos cambios. Se puede configurar un filtro de eventos en los desencadenadores de Lambda para filtrar los eventos que no sean pertinentes para el caso de uso. En este caso, podemos utilizar un filtro para desencadenar Lambda solo cuando se agregue un nuevo comentario y enviar una notificación a los ID de correo electrónico pertinentes, que pueden obtenerse de AWS Secrets Manager o de cualquier otro almacén de credenciales.

Ejecutar consultas analíticas sobre los datos semanales

DynamoDB es adecuado para cargas de trabajo centradas principalmente en el procesamiento transaccional en línea (OLTP). Para el otro 10 % a 20 % de patrones de acceso con requisitos analíticos, los datos pueden exportarse a S3 con la característica administrada Exportar a Amazon S3 sin impacto en el tráfico en directo de la tabla de DynamoDB. Examinemos este flujo de trabajo:

Amazon EventBridge puede utilizarse para desencadenar AWS Lambda de forma programada: permite configurar una expresión cron para que la invocación de Lambda tenga lugar periódicamente. Lambda puede invocar la llamada a la API `ExportToS3` y almacenar los datos de DynamoDB en S3. A continuación, un motor SQL como Amazon Athena puede acceder a estos datos de S3 para ejecutar consultas analíticas en los datos de DynamoDB sin que se vea afectada la carga de trabajo transaccional en directo de la tabla. Un ejemplo de consulta de Athena para hallar el número de reclamaciones por nivel de gravedad tendría el siguiente aspecto:

```
SELECT Item.severity.S as "Severity", COUNT(Item) as "Count"
FROM "complaint_management"."data"
WHERE NOT Item.severity.S = ''
GROUP BY Item.severity.S ;
```

Esto da como resultado la siguiente consulta de Athena:

Results (3)

Search rows

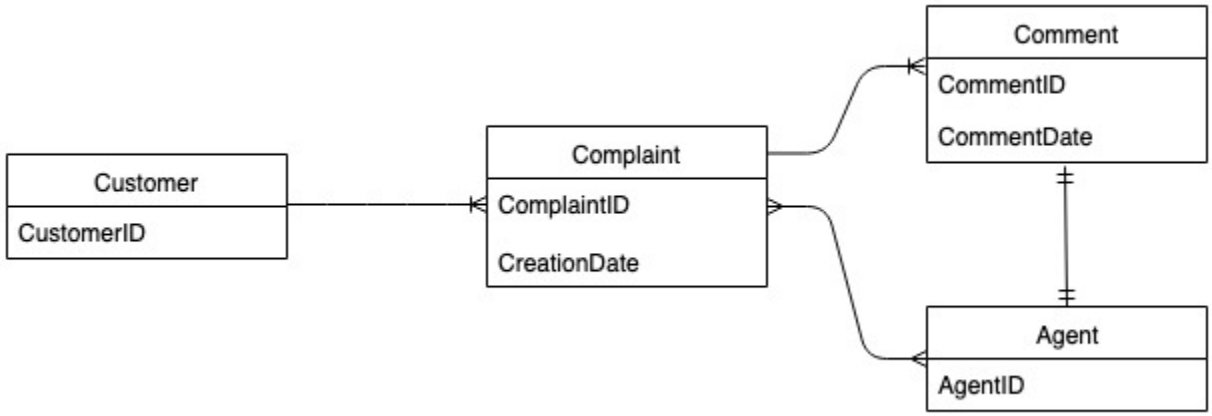
#	Severity	Count
1	P3	1
2	P2	2
3	P1	1

Archivar datos de más de tres años

Puede aprovechar la característica Tiempo de vida (TTL) de DynamoDB para eliminar datos obsoletos de su tabla de DynamoDB sin costo adicional (excepto en el caso de las réplicas de tablas globales para la versión 2019.11.21 (actual), donde las eliminaciones TTL replicadas a otras regiones consumen capacidad de escritura). Estos datos aparecen y se pueden consumir desde DynamoDB Streams para archivarse en Amazon S3. El flujo de trabajo para este requisito es el siguiente:

Diagrama de relaciones entre entidades del sistema de administración de reclamaciones

Este es el diagrama de relaciones entre entidades (ERD) que utilizaremos para el diseño del esquema del sistema de administración de reclamaciones.



Patrones de acceso al sistema de administración de reclamaciones

Estos son los patrones de acceso que tendremos en cuenta para el diseño del esquema de administración de reclamaciones.

1. createComplaint
2. updateComplaint
3. updateSeveritybyComplaintID

4. getComplaintByComplaintID
5. addCommentByComplaintID
6. getAllCommentsByComplaintID
7. getLatestCommentByComplaintID
8. getAComplaintbyCustomerIDAndComplaintID
9. getAllComplaintsByCustomerID
10. escalateComplaintByComplaintID
11. getAllEscalatedComplaints
12. getEscalatedComplaintsByAgentID (orden de más reciente a más antiguo)
13. getCommentsByAgentID (entre dos fechas)

Evolución del diseño del esquema del sistema de administración de reclamaciones

Al tratarse de un sistema de administración de reclamaciones, la mayoría de los patrones de acceso giran en torno a la reclamación como entidad principal. El valor de `ComplaintID`, al ser altamente cardinal, garantizará una distribución uniforme de los datos en las particiones subyacentes y es también el criterio de búsqueda más común para nuestros patrones de acceso identificados. Por lo tanto, `ComplaintID` es una buena candidata a clave de partición en este conjunto de datos.

Paso 1: Abordar los patrones de acceso 1 (**`createComplaint`**), 2 (**`updateComplaint`**), 3 (**`updateSeveritybyComplaintID`**) y 4 (**`getComplaintByComplaintID`**)

Podemos utilizar una clave de clasificación genérica valorada como "metadatos" (o "AA") para almacenar información específica de la reclamación como `CustomerID`, `State`, `Severity` y `CreationDate`. Utilizamos operaciones singleton con `PK=ComplaintID` y `SK="metadata"` para hacer lo siguiente:

1. `PutItem` para crear una nueva reclamación
2. `UpdateItem` para actualizar la gravedad u otros campos de los metadatos de la reclamación
3. `GetItem` para obtener los metadatos de la reclamación

Primary key		Attributes				
Partition key: PK	Sort key: SK					
Complaint1321	metadata	customer_id	current_state	creation_time	severity	complaint_description
		custXYZ	assigned	2023-05-10T15:58:00	P2	<Complaint Description>

Paso 2: Abordar el patrón de acceso 5 (**`addCommentByComplaintID`**)

Este patrón de acceso requiere un modelo de relación de uno a varios entre una reclamación y los comentarios sobre ella. Aquí utilizaremos la técnica de partición vertical para utilizar una clave de clasificación y crear una colección de elementos con distintos tipos de datos. Si observamos los patrones de acceso 6 (`getAllCommentsByComplaintID`) y 7 (`getLatestCommentByComplaintID`), sabremos que los comentarios deberán ordenarse por tiempo. También podemos tener varios comentarios que lleguen al mismo tiempo, por lo que podemos utilizar la técnica de clave de clasificación compuesta para agregar la hora y `CommentID` en el atributo de clave de clasificación.

Otras opciones para tratar estas posibles colisiones de comentarios serían aumentar la granularidad para la marca de tiempo o agregar un número incremental como sufijo en lugar de utilizar `Comment_ID`. En este caso, antepondremos el prefijo “comm#” al valor de la clave de clasificación de los elementos correspondientes a comentarios para permitir operaciones basadas en intervalos.

También tenemos que asegurarnos de que `currentState` en los metadatos de la reclamación refleja el estado cuando se agrega un nuevo comentario. Agregar un comentario puede indicar que la reclamación se ha asignado a un agente o que se ha resuelto, etc. Para agrupar la adición de comentarios y la actualización del estado actual en los metadatos de la reclamación, de forma que todo sea posible, utilizaremos la API TransactWriteItems. El estado de la tabla resultante tiene ahora este aspecto:

Primary key		Attributes				
Partition key: PK	**Sort key: SK**					
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID
		comm3	2023-05-10T16:00:00	investigating	<Comment text>	AgentB
	metadata	customer_id	current_state	creation_time	severity	complaint_description
		custXYZ	investigating	2023-05-10T15:58:00	P2	<Complaint Description>

Vamos a agregar algunos datos más en la tabla y también agregar `ComplaintID` como un campo separado de nuestro PK para preparar el modelo para el futuro en caso de que necesitemos índices adicionales en `ComplaintID`. Tenga en cuenta también que algunos comentarios pueden tener archivos adjuntos que almacenaremos en Amazon Simple Storage Service y solo mantendremos sus referencias o URL en DynamoDB. Se recomienda mantener la base de datos transaccional lo más reducida posible para optimizar los costos y el rendimiento. Los datos ahora tienen este aspecto:

Primary key		Attributes					
Partition key: PK	**Sort key: SK**						
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Paso 3: Abordar patrones de acceso 6 (**`getAllCommentsByComplaintID`**) y 7 (**`getLatestCommentByComplaintID`**)

Para obtener todos los comentarios de una reclamación, podemos utilizar la operación `query` con la condición `begins_with` en la clave de clasificación. En lugar de consumir capacidad de lectura adicional para leer la entrada de metadatos y, a continuación, tener la sobrecarga de filtrar los resultados relevantes, tener una condición de clave de clasificación como esta nos ayuda a leer solo lo que necesitamos. Por ejemplo, una operación de consulta con `PK=Complaint123` y SK begins_with `comm#` devolvería lo siguiente al mismo tiempo que omitiría la entrada de metadatos:

Primary key		Attributes					
Partition key: PK	**Sort key: SK**						
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Puesto que necesitamos el último comentario de una reclamación en el patrón 7 (`getLatestCommentByComplaintID`), vamos a utilizar dos parámetros de consulta adicionales:

1. `ScanIndexForward` debe configurarse en False para que los resultados se ordenen en orden descendente
2. `Limit` debe configurarse en 1 para obtener el último comentario (solo uno)

De forma similar al patrón de acceso 6 (`getAllCommentsByComplaintID`), omitimos la entrada de metadatos mediante `begins_with comm#` como la condición de clave de clasificación. Ahora, puede realizar el patrón de acceso 7 en este diseño mediante la operación de consulta con `PK=Complaint123` y `SK=begins_with comm#`, `ScanIndexForward=False` y `Limit` 1. Como resultado se devolverá el siguiente elemento objetivo:

Partition key: PK	Sort key: SK	Attributes					
	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
Complaint123	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Agreguemos más datos ficticios a la tabla.

Primary key		Attributes					
Partition key: PK	**Sort key: SK**						
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA	
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID	
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB	
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>
Complaint1444	comm#2022-12-31T19:32:00#comm4	comm_id	comm_date	complaint_state	comm_text		
		comm4	2022-12-31T19:32:00	waiting	<comm text>		
	comm#2022-12-31T19:40:00#comm5	comm_id	comm_date	complaint_state	comm_text	attachments	agentID
		comm5	2022-12-31T19:40:00	assigned	<comm text>	["s3://URL_for_attachment1"]	AgentC
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>
Complaint0987	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		custXYZ	Complaint0987	assigned	2023-06-10T12:30:08	P3	<description text>

Paso 4: Abordar patrones de acceso 8 (**`getAComplaintbyCustomerIDAndComplaintID`**) y 9 (**`getAllComplaintsByCustomerID`**)

El acceso a los patrones 8 (`getAComplaintbyCustomerIDAndComplaintID`) y 9 (`getAllComplaintsByCustomerID`) presenta un nuevo criterio de búsqueda: `CustomerID`. Recuperarlo de la tabla existente requiere un proceso `Scan` costoso de lectura de todos los datos y, a continuación, filtrar los elementos relevantes para `CustomerID` en cuestión.

Podemos hacer que esta búsqueda sea más eficaz mediante la creación de un índice secundario global (GSI) con `CustomerID` como clave de partición. Teniendo en cuenta la relación de uno a varios entre cliente y reclamaciones, así como el patrón de acceso 9 (`getAllComplaintsByCustomerID`), `ComplaintID` sería el candidato adecuado para la clave de clasificación.

Los datos del GSI tendrían este aspecto:

Primary key		**Attributes**					
Partition key: customer_id	**Sort key: complaint_id**						
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

Un ejemplo de consulta en este GSI para el patrón de acceso 8 (`getAComplaintbyCustomerIDAndComplaintID`) sería: `customer_id=custXYZ`, `sort key=Complaint1321`. El resultado sería:

Primary key		Attributes					
Partition key: customer_id	**Sort key: complaint_id**						
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

Para obtener todas las reclamaciones de un cliente para el patrón de acceso 9 (`getAllComplaintsByCustomerID`), la consulta del GSI sería: `customer_id=custXYZ` como condición de clave de partición. El resultado sería:

Primary key		Attributes					
Partition key: customer_id	**Sort key: complaint_id**						
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>

Paso 5: Abordar el patrón de acceso 10 (**`escalateComplaintByComplaintID`**)

Este acceso introduce el aspecto de escalado. Para escalar una reclamación, podemos utilizar `UpdateItem` para agregar atributos como `escalated_to` y `escalation_time` al elemento de metadatos de reclamación existente. DynamoDB proporciona un diseño de esquema flexible,

lo que significa que un conjunto de atributos no clave puede ser uniforme o discreto en diferentes elementos. Consulte a continuación un ejemplo:

`UpdateItem with PK=Complaint1444, SK=metadata`

Complaint1444	comm#2022-12-31T19:32:00#comm4	comm_id	comm_date	complaint_state	comm_text				
		comm4	2022-12-31T19:32:00	waiting	<comm text>				
	comm#2022-12-31T19:40:00#comm5	comm_id	comm_date	complaint_state	comm_text	attachments	agentID		
		comm5	2022-12-31T19:40:00	assigned	<comm text>	["s3://URL_for_attachment1"]	AgentC		
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07

Paso 6: Abordar patrones de acceso 11 (**`getAllEscalatedComplaints`**) y 12 (**`getEscalatedComplaintsByAgentID`**)

De todo el conjunto de datos, solo se espera que se escalen unas pocas reclamaciones. Por lo tanto, la creación de un índice sobre los atributos relacionados con el escalado conduciría a búsquedas eficientes, así como a un almacenamiento de GSI rentable. Podemos hacerlo aprovechando la técnica del índice disperso. El GSI con clave de partición como `escalated_to` y clave de clasificación como `escalation_time` tendría el siguiente aspecto:

Primary key		Attributes							
Partition key: escalated_to	Sort key: escalation_time								
AgentB	2023-01-03T04:00:07	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>
	2023-05-15T14:00:00	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Para obtener todas las reclamaciones escaladas para el patrón de acceso 11 (`getAllEscalatedComplaints`), simplemente escaneamos este GSI. Tenga en cuenta que esta escaneo será eficaz y rentable debido al tamaño del GSI. Para obtener reclamaciones escaladas para un agente específico (patrón de acceso 12 [`getEscalatedComplaintsByAgentID`]), la clave de partición sería `escalated_to=agentID` y establecemos `ScanIndexForward aFalse` para ordenar de más reciente a más antiguo.

Paso 7: Abordar el patrón de acceso 13 (**`getCommentsByAgentID`**)

Para el último patrón de acceso, necesitamos realizar una búsqueda por una nueva dimensión: `AgentID`. También necesitamos una ordenación basada en el tiempo para leer los

comentarios entre dos fechas, así que creamos un GSI con `agent_id` como la clave de partición y `comm_date` como la clave de clasificación. Los datos de este GSI tendrán el siguiente aspecto:

Primary key		Attributes					
Partition key: agentID	**Sort key: comm_date**						
AgentA	2023-04-30T12:00:24	PK	SK	comm_id	complaint_state	comm_text	
		Complaint123	comm#2023-04-30T12:00:24#comm1	comm1	investigating	<comm text>	
	2023-04-30T12:35:54	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint123	comm#2023-04-30T12:35:54#comm2	comm2	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]
AgentB	2023-05-10T16:00:00	PK	SK	comm_id	complaint_state	comm_text	
		Complaint1321	comm#2023-05-10T16:00:00#comm3	comm3	investigating	<comm text>	
AgentC	2022-12-31T19:40:00	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint1444	comm#2022-12-31T19:40:00#comm5	comm5	assigned	<comm text>	["s3://URL_for_attachment1"]

Un ejemplo de consulta en este GSI sería `partition key agentID=AgentA` y `sort key=comm_date between (2023-04-30T12:30:00, 2023-05-01T09:00:00)`, cuyo resultado es:

Primary key		Attributes					
Partition key: agentID	**Sort key: comm_date**						
AgentA	2023-04-30T12:00:24	PK	SK	comm_id	complaint_state	comm_text	
		Complaint123	comm#2023-04-30T12:00:24#comm1	comm1	investigating	<comm text>	
	2023-04-30T12:35:54	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint123	comm#2023-04-30T12:35:54#comm2	comm2	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]
AgentB	2023-05-10T16:00:00	PK	SK	comm_id	complaint_state	comm_text	
		Complaint1321	comm#2023-05-10T16:00:00#comm3	comm3	investigating	<comm text>	
AgentC	2022-12-31T19:40:00	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint1444	comm#2022-12-31T19:40:00#comm5	comm5	assigned	<comm text>	["s3://URL_for_attachment1"]

En la tabla siguiente se resumen todos los patrones de acceso y cómo los aborda el diseño del esquema:

Patrón de acceso	Tabla base/ GSI/LSI	Operación	Valor de clave de partición	Valor de la clave de clasificación	Otras condiciones/ filtros
createComplaint	Tabla base	PutItem	PK=complaint_id	SK=metadata	
updateComplaint	Tabla base	UpdateItem	PK=complaint_id	SK=metadata	
updateSeveritybyComplaintID	Tabla base	UpdateItem	PK=complaint_id	SK=metadata	
getComplaintByComplaintID	Tabla de base	GetItem	PK=complaint_id	SK=metadata	
addCommentByComplaintID	Tabla base	TransactWriteItems	PK=complaint_id	SK=metadata, SK=comm#comm_date#comm_id	
getAllCommentsByComplaintID	Tabla base	Consultar	PK=complaint_id	SK begins_with “comm#”	
getLatestCommentByComplaintID	Tabla base	Consultar	PK=complaint_id	SK begins_with “comm#”	scan_index_forward=False, Limit 1
getAComplaintbyCustomerIDAndComplaintID	Customer_complaint_GSI	Consultar	customer_id=customer_id	complaint_id = complaint_id	

Patrón de acceso	Tabla base/ GSI/LSI	Operación	Valor de clave de partición	Valor de la clave de clasificación	Otras condiciones/ filtros
getAllComplaintsByCustomerID	Customer_complaint_GSI	Consultar	customer_id=customer_id	N/A	
escalateComplaintByComplaintID	Tabla base	UpdateItem	PK=complaint_id	SK=metadata	
getAllEscalatedComplaints	Escalations_GSI	Examen	N/A	N/A	
getEscalatedComplaintsByAgentID (orden de más reciente a más antiguo)	Escalations_GSI	Consultar	escalated_to=agent_id	N/A	scan_index_forward=False
getCommentsByAgentID (entre dos fechas)	Agents_Comments_GSI	Consultar	agent_id=agent_id	SK entre (fecha1, fecha2)	

Esquema final del sistema de administración de reclamaciones

Estos son los diseños finales del esquema. Para descargar este diseño de esquema como un archivo JSON, consulte los ejemplos de DynamoDB en GitHub.

Tabla base

Primary key		Attributes							
Partition key: PK	**Sort key: SK**								
Complaint123	comm#2023-04-30T12:00:24#comm1	comm_id	comm_date	complaint_state	comm_text	agentID			
		comm1	2023-04-30T12:00:24	investigating	<comm text>	AgentA			
	comm#2023-04-30T12:35:54#comm2	comm_id	comm_date	complaint_state	comm_text	attachments	agentID		
		comm2	2023-04-30T12:35:54	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]	AgentA		
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description		
		custABC	Complaint123	resolved	2023-04-30T12:00:00	P2	<description text>		
Complaint1321	comm#2023-05-10T16:00:00#comm3	comm_id	comm_date	complaint_state	comm_text	agentID			
		comm3	2023-05-10T16:00:00	investigating	<comm text>	AgentB			
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>	AgentB	2023-05-15T14:00:00
Complaint1444	comm#2022-12-31T19:32:00#comm4	comm_id	comm_date	complaint_state	comm_text				
		comm4	2022-12-31T19:32:00	waiting	<comm text>				
	comm#2022-12-31T19:40:00#comm5	comm_id	comm_date	complaint_state	comm_text	attachments	agentID		
		comm5	2022-12-31T19:40:00	assigned	<comm text>	["s3://URL_for_attachment1"]	AgentC		
	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07
Complaint0987	metadata	customer_id	complaint_id	current_state	creation_time	severity	complaint_description		
		custXYZ	Complaint0987	assigned	2023-06-10T12:30:08	P3	<description text>		

Customer_Complaint_GSI

Primary key		Attributes							
Partition key: customer_id	**Sort key: complaint_id**								
custABC	Complaint123	PK	SK	current_state	creation_time	severity	complaint_description		
		Complaint123	metadata	resolved	2023-04-30T12:00:00	P2	<description text>		
custXY32	Complaint1444	PK	SK	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		Complaint1444	metadata	assigned	2022-12-31T19:39:57	P1	<description text>	AgentB	2023-01-03T04:00:07
custXYZ	Complaint0987	PK	SK	current_state	creation_time	severity	complaint_description		
		Complaint0987	metadata	assigned	2023-06-10T12:30:08	P3	<description text>		
	Complaint1321	PK	SK	current_state	creation_time	severity	complaint_description	escalated_to	escalation_time
		Complaint1321	metadata	investigating	2023-05-10T15:58:00	P2	<descr_text>	AgentB	2023-05-15T14:00:00

Escalations_GSI

Primary key		Attributes							
Partition key: escalated_to	Sort key: escalation_time								
AgentB	2023-01-03T04:00:07	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1444	metadata	custXY32	Complaint1444	assigned	2022-12-31T19:39:57	P1	<description text>
	2023-05-15T14:00:00	PK	SK	customer_id	complaint_id	current_state	creation_time	severity	complaint_description
		Complaint1321	metadata	custXYZ	Complaint1321	investigating	2023-05-10T15:58:00	P2	<descr_text>

Agents_Comments_GSI

Primary key		Attributes					
Partition key: agentID	Sort key: comm_date						
AgentA	2023-04-30T12:00:24	PK	SK	comm_id	complaint_state	comm_text	
		Complaint123	comm#2023-04-30T12:00:24#comm1	comm1	investigating	<comm text>	
	2023-04-30T12:35:54	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint123	comm#2023-04-30T12:35:54#comm2	comm2	resolved	<comm text>	["s3://URL_for_attachment1","s3://URL_for_attachment2"]
AgentB	2023-05-10T16:00:00	PK	SK	comm_id	complaint_state	comm_text	
		Complaint1321	comm#2023-05-10T16:00:00#comm3	comm3	investigating	<comm text>	
AgentC	2022-12-31T19:40:00	PK	SK	comm_id	complaint_state	comm_text	attachments
		Complaint1444	comm#2022-12-31T19:40:00#comm5	comm5	assigned	<comm text>	["s3://URL_for_attachment1"]

Uso de NoSQL Workbench con este diseño de esquema

Puede importar este esquema final en NoSQL Workbench, una herramienta visual que proporciona características de modelado de datos, visualización de datos y desarrollo de consultas para DynamoDB, a fin de explorar y editar más a fondo el nuevo proyecto. Para comenzar, siga estos pasos:

1. Descargue NoSQL Workbench. Para obtener más información, consulte the section called “Descargar”.
2. Descargue el archivo de esquema JSON que se muestra anteriormente, que ya está en el formato de modelo NoSQL Workbench.
3. Importe el archivo de esquema JSON en NoSQL Workbench. Para obtener más información, consulte the section called “Importación de un modelo existente”.
4. Una vez que haya importado en NOSQL Workbench, podrá editar el modelo de datos. Para obtener más información, consulte the section called “Edición de un modelo existente”.
5. Para visualizar el modelo de datos, agregar datos de ejemplo o importar datos de ejemplo de un archivo CSV, utilice la característica Visualizador de datos de NoSQL Workbench.

Diseño del esquema de pagos periódicos en DynamoDB

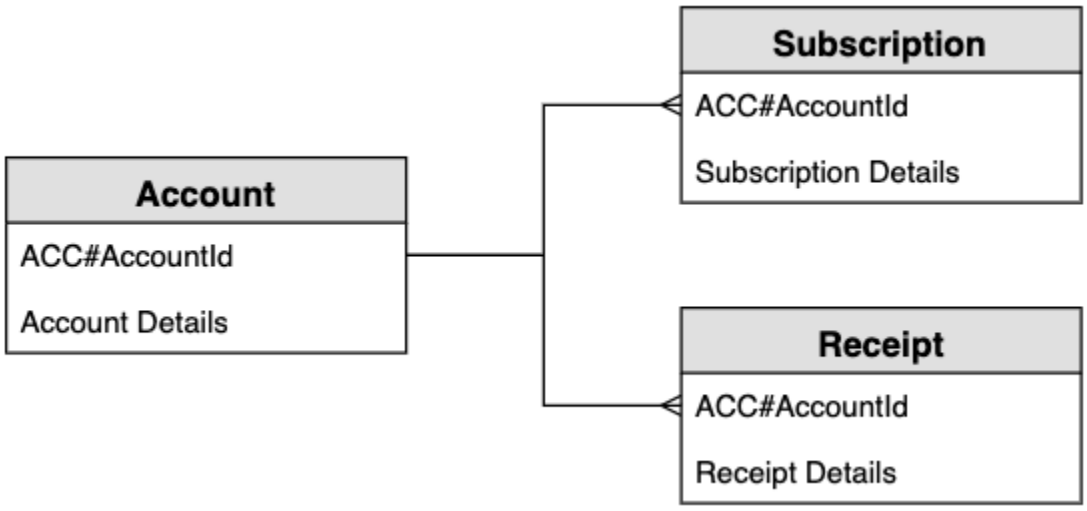
Caso de uso empresarial de pagos periódicos

En este caso de uso se habla de utilizar DynamoDB para implementar un sistema de pagos periódicos. El modelo de datos tiene las siguientes entidades: cuentas, suscripciones y recibos. Los detalles específicos para nuestro caso de uso son los siguientes:

- Cada cuenta puede tener varias suscripciones
- La suscripción tiene una `NextPaymentDate` en la que se debe procesar el siguiente pago y una `NextReminderDate` en la que se envía un recordatorio por correo electrónico al cliente
- Existe un elemento para la suscripción que se almacena y actualiza cuando se procesa el pago (el tamaño medio del elemento es de alrededor de 1 KB y el rendimiento depende del número de cuentas y suscripciones).
- El procesador de pagos también creará un recibo como parte del proceso que se almacena en la tabla y se establece a expirar después de un periodo de tiempo mediante un atributo TTL

Diagrama de relación entre entidades de pagos periódicos

Este es el diagrama de relaciones entre entidades (ERD) que utilizaremos para el diseño del esquema del sistema de pagos periódicos.



Patrones de acceso al sistema de pagos periódicos

Estos son los patrones de acceso que tendremos en cuenta para el diseño del esquema del sistema de pagos periódicos.

1. `createSubscription`
2. `createReceipt`
3. `updateSubscription`
4. `getDueRemindersByDate`
5. `getDuePaymentsByDate`
6. `getSubscriptionsByAccount`
7. `getReceiptsByAccount`

Diseño del esquema de pagos periódicos

Los nombres genéricos `PK` y `SK` se utilizan para atributos de clave que permiten almacenar distintos tipos de entidades en la misma tabla, como las entidades de cuenta, suscripción y recibo. En primer lugar, el usuario crea una suscripción, por la que se compromete a pagar una cantidad el mismo día cada mes a cambio de un producto. Puede elegir qué día del mes se procesa el pago. También se enviará un recordatorio antes de que se procese el pago. La aplicación funciona mediante dos trabajos por lotes que se ejecutan cada día: un trabajo por lotes envía los recordatorios que vencen ese día y el otro procesa los pagos que vencen ese día.

Paso 1: Abordar el patrón de acceso 1 (**`createSubscription`**)

El patrón de acceso 1 (`createSubscription`) se utiliza para crear inicialmente la suscripción y se establecen los detalles, como `SKU`, `NextPaymentDate`, `NextReminderDate` y `PaymentDetails`. En este paso se muestra el estado de la tabla para una sola cuenta con una suscripción. Puede haber varias suscripciones en la colección de elementos, por lo que se trata de una relación de uno a varios.

Primary key		Attributes									
Partition key: PK	**Sort key: SK**										
		Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
ACC#123	SUB#123#SKU#999	s@s.com	28	12.99	1970-01-01T00:00:00.000Z	2023-05-28	1970-01-01T00:00:00.000Z	2023-05-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

Paso 2: Abordar patrones de acceso 2 (**`createReceipt`**) y 3 (**`updateSubscription`**)

Se utiliza el patrón de acceso 2 (`createReceipt`) para crear el elemento de recibo. Una vez procesado el pago cada mes, el procesador de pagos volverá a escribir un recibo en la tabla base. Puede haber varios recibos en la colección de elementos, por lo que se trata de una relación de uno a varios. El procesador de pagos también actualizará el elemento de suscripción (patrón de acceso 3

[updateSubscription]) para actualizar para la NextReminderDate o la NextPaymentDate del mes siguiente.

Primary key		Attributes									
Partition key: PK	Sort key: SK										
ACC#123	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

Paso 3: Abordar el patrón de acceso 4 (**getDueRemindersByDate**)

La aplicación procesa los recordatorios de pago por lotes correspondientes al día en curso. Por lo tanto, la aplicación necesita acceder a las suscripciones en una dimensión diferente: fecha en lugar de cuenta. Este es un buen caso de uso para un índice secundario global (GSI). En este paso agregamos el índice GSI-1, que utiliza la NextReminderDate como clave de partición de GSI. No necesitamos replicar todos los elementos. Este GSI es un índice disperso y los elementos de entrada no están replicados. Tampoco necesitamos proyectar todos los atributos, solo necesitamos incluir un subconjunto de ellos. En la imagen siguiente se muestra el esquema de GSI-1 y se proporciona la información necesaria para que la aplicación envíe el correo electrónico de recordatorio.

Primary key		Attributes				
Partition key: NextReminderDate	Sort key: LastReminderDate					
2023-06-21	2023-05-21T14:15:39.247Z	SK	PK	SKU	Email	NextPaymentDate
		SUB#123#SKU#999	ACC#123	999	s@s.com	2023-06-28

Paso 4: Abordar el patrón de acceso 5 (**getDuePaymentsByDate**)

La aplicación procesa los pagos en lotes para el día en curso del mismo modo que lo hace con los recordatorios. Agregamos GSI-2 en este paso y utiliza la NextPaymentDate como la clave de partición de GSI. No necesitamos replicar todos los elementos. Este GSI es un índice disperso, ya que los elementos de entrada no están replicados. En la imagen siguiente se muestra el esquema de GSI-2.

Primary key		Attributes						
Partition key: NextPaymentDate	Sort key: LastPaymentDate							
2023-06-28	2023-05-18T14:15:39.247Z	PK	SK	Email	PaymentDay	PaymentAmount	SKU	PaymentDetails
		ACC#123	SUB#123#SKU#999	s@s.com	28	12.99	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}

Paso 5: Abordar patrones de acceso 6 (**getSubscriptionsByAccount**) y 7 (**getReceiptsByAccount**)

La aplicación puede recuperar todas las suscripciones de una cuenta mediante una consulta en la tabla base que se dirige al identificador de la cuenta (el PK) y utiliza el operador de intervalo para obtener todos los elementos en los que el SK comienza por “SUB#”. La aplicación también puede utilizar la misma estructura de consulta para recuperar todos los recibos empleando un operador de intervalo para obtener todos los elementos en los que el SK comience por “REC#”. Esto nos permite satisfacer los patrones de acceso 6 (`getSubscriptionsByAccount`) y 7 (`getReceiptsByAccount`). La aplicación utiliza estos patrones de acceso para que el usuario pueda ver sus suscripciones actuales y sus recibos anteriores de los últimos seis meses. No hay ningún cambio en el esquema de la tabla en este paso y podemos ver a continuación cómo nos dirigimos solo a los elementos de suscripción en el patrón de acceso 6 (`getSubscriptionsByAccount`).

Primary key		Attributes									
Partition key: PK	Sort key: SK										
	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
ACC#123		Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
	SUB#123#SKU#999	s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

En la tabla siguiente se resumen todos los patrones de acceso y cómo los aborda el diseño del esquema:

Patrón de acceso	Tabla base/GSI/LSI	Operación	Valor de clave de partición	Valor de la clave de clasificación
createSubscription	Tabla base	PutItem	ACC#account_id	SUB#<SUBID>#SKU<SKUID>
createReceipt	Tabla base	PutItem	ACC#account_id	REC#<ReceiptDate>#SKU<SKUID>
updateSubscription	Tabla base	UpdateItem	ACC#account_id	SUB#<SUBID>#SKU<SKUID>

Patrón de acceso	Tabla base/GSI/ LSI	Operación	Valor de clave de partición	Valor de la clave de clasificación
getDueRem indersByDate	GSI-1	Consultar	<NextRemi nderDate>	
getDuePay mentsByDate	GSI-2	Consultar	<NextPaym entDate>	
getSubscr iptionsByAccount	Tabla base	Consultar	ACC#account_id	SK begins_with “SUB#”
getReceip tsByAccount	Tabla base	Consultar	ACC#account_id	SK begins_with “REC#”

Esquema final de pagos periódicos

Estos son los diseños finales del esquema. Para descargar este diseño de esquema como un archivo JSON, consulte los ejemplos de DynamoDB en GitHub.

Tabla base

Primary key		Attributes									
Partition key: PK	Sort key: SK										
ACC#123	REC#12023-05-28T14:15:39.24#SKU#999	Email	SKU	ProcessedDate	ProcessedAmount	TTL					
		s@s.com	999	2023-05-28T14:15:39.247Z	12.99	1700318200					
	SUB#123#SKU#999	Email	PaymentDay	PaymentAmount	LastPaymentDate	NextPaymentDate	LastReminderDate	NextReminderDate	SKU	PaymentDetails	CreatedDate
		s@s.com	28	12.99	2023-05-18T14:15:39.247Z	2023-06-28	2023-05-21T14:15:39.247Z	2023-06-21	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}	2023-05-18T09:41:25.856Z

GSI-1

Primary key		Attributes				
Partition key: NextReminderDate	Sort key: LastReminderDate	SK	PK	SKU	Email	NextPaymentDate
2023-06-21	2023-05-21T14:15:39.247Z	SUB#123#SKU#999	ACC#123	999	s@s.com	2023-06-28

GSI-2

Primary key		Attributes						
Partition key: NextPaymentDate	Sort key: LastPaymentDate	PK	SK	Email	PaymentDay	PaymentAmount	SKU	PaymentDetails
2023-06-28	2023-05-18T14:15:39.247Z	ACC#123	SUB#123#SKU#999	s@s.com	28	12.99	999	{"default-card":{"S":"1234123412341234"},"default-address":{"S":"12 Bridge Street, Birmingham, B12 7ST"}}

Uso de NoSQL Workbench con este diseño de esquema

Puede importar este esquema final en NoSQL Workbench, una herramienta visual que proporciona características de modelado de datos, visualización de datos y desarrollo de consultas para DynamoDB, a fin de explorar y editar más a fondo el nuevo proyecto. Para comenzar, siga estos pasos:

1. Descargue NoSQL Workbench. Para obtener más información, consulte the section called “Descargar”.
2. Descargue el archivo de esquema JSON que se muestra anteriormente, que ya está en el formato de modelo NoSQL Workbench.
3. Importe el archivo de esquema JSON en NoSQL Workbench. Para obtener más información, consulte the section called “Importación de un modelo existente”.
4. Una vez que haya importado en NOSQL Workbench, podrá editar el modelo de datos. Para obtener más información, consulte the section called “Edición de un modelo existente”.
5. Para visualizar el modelo de datos, agregar datos de ejemplo o importar datos de ejemplo de un archivo CSV, utilice la característica Visualizador de datos de NoSQL Workbench.

Monitoreo de las actualizaciones de estado de los dispositivos en DynamoDB

En este caso de uso se habla de utilizar DynamoDB para monitorear las actualizaciones del estado del dispositivo (o los cambios en el estado del dispositivo) en DynamoDB.

Caso de uso

En los casos de uso de IoT (una fábrica inteligente, por ejemplo), los operadores deben monitorear muchos dispositivos y estos envían periódicamente su estado o sus registros a un sistema de monitoreo. Cuando hay un problema con un dispositivo, el estado del mismo cambia de normal a advertencia. Existen diferentes niveles o estados de registro en función de la gravedad y el tipo de comportamiento anómalo del dispositivo. A continuación, el sistema asigna a un operario para que compruebe el dispositivo y este puede escalar el problema a su supervisor si es necesario.

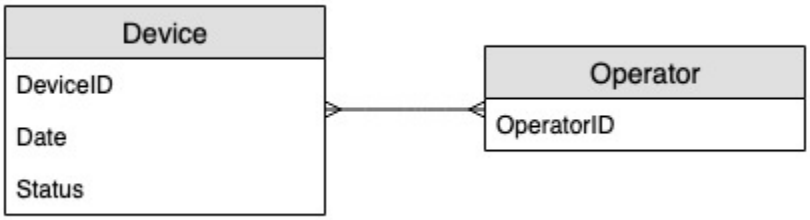
Algunos patrones de acceso típicos de este sistema son:

- Crear una entrada de registro para un dispositivo

- Obtener todos los registros para un estado de dispositivo específico mostrando primero los registros más recientes
- Obtener todos los registros de un operador dado entre dos fechas
- Obtener todos los registros escalados de un supervisor determinado
- Obtener todos los registros escalados con un estado de dispositivo específico para un supervisor determinado
- Obtener todos los registros escalados con un estado de dispositivo específico para un supervisor determinado en una fecha concreta

Diagrama de relaciones entre entidades

Este es el diagrama de relaciones entre entidades (ERD) que utilizaremos para monitorear las actualizaciones del estado de los dispositivos.



Patrones de acceso

Estos son los patrones de acceso que tendremos en cuenta para monitorear las actualizaciones de estado de los dispositivos.

1. `createLogEntryForSpecificDevice`
2. `getLogsForSpecificDevice`
3. `getWarningLogsForSpecificDevice`
4. `getLogsForOperatorBetweenTwoDates`
5. `getEscalatedLogsForSupervisor`
6. `getEscalatedLogsWithSpecificStatusForSupervisor`
7. `getEscalatedLogsWithSpecificStatusForSupervisorForDate`

Evolución del diseño de esquema

Paso 1: Abordar patrones de acceso 1 (**`createLogEntryForSpecificDevice`**) y 2 (**`getLogsForSpecificDevice`**)

La unidad de escala para un sistema de seguimiento de dispositivos serían los dispositivos individuales. En este sistema, un `deviceID` identifica de forma única a un dispositivo. Esto convierte a `deviceID` en un buen candidato para la clave de partición. Cada dispositivo envía información al sistema de seguimiento periódicamente (por ejemplo, cada cinco minutos más o menos). Esta ordenación convierte a la fecha en un criterio lógico de clasificación y, por tanto, en la clave de clasificación. Los datos de muestra en este caso tendrían este aspecto:

Primary key		Attributes
Partition key: DeviceID	Sort key: State#Date	
d#12345	2020-04-24T14:40:00	State
		WARNING1
	2020-04-24T14:45:00	State
		WARNING1
	2020-04-24T14:50:00	State
		WARNING1
	2020-04-24T14:55:00	State
		NORMAL
d#54321	2020-04-11T05:50:00	State
		WARNING3
	2020-04-11T05:55:00	State
		WARNING3
	2020-04-11T06:00:00	State
		NORMAL
	2020-04-11T09:25:00	State
		WARNING2
	2020-04-11T09:30:00	State
		NORMAL
d#11223	2020-04-27T16:10:00	State
		WARNING4
	2020-04-27T16:15:00	State
		WARNING4

Para recuperar las entradas de registro de un dispositivo concreto, podemos realizar una operación de consulta con clave de partición `DeviceID="d#12345"`.

Paso 2: Abordar el patrón de acceso 3 (**`getWarningLogsForSpecificDevice`**)

Dado que `State` es un atributo no clave, abordar el patrón de acceso 3 con el esquema actual requeriría una expresión de filtro. En DynamoDB, las expresiones de filtro se aplican después de leer los datos mediante expresiones de condición clave. Por ejemplo, si quisiéramos obtener los registros de advertencias de `d#12345`, la operación de consulta con clave de partición `DeviceID="d#12345"` leerá cuatro elementos de la tabla anterior y, a continuación, filtrará el único elemento sin el estado de advertencia. Este enfoque no es eficiente a escala. Una expresión de filtro puede ser una buena forma de excluir elementos consultados si la proporción de elementos excluidos es baja o la consulta se realiza con poca frecuencia. No obstante, para los casos en los que se recuperan muchos elementos de una tabla y la mayoría de los elementos se filtran, podemos seguir evolucionando el diseño de nuestra tabla para que funcione de forma más eficiente.

Cambiemos la forma de gestionar este patrón de acceso mediante claves de clasificación compuestas. Puede importar datos de muestra de DeviceStateLog_3.json, donde la clave de clasificación se cambia a `State#Date`. Esta clave de clasificación es la composición de los atributos `State`, `#` y `Date`. En este ejemplo, `#` se utiliza como delimitador. Los datos tienen ahora este aspecto:

Primary key	
Partition key: DeviceID	**Sort key: State#Date**
d#12345	NORMAL#2020-04-24T14:55:00
	WARNING1#2020-04-24T14:40:00
	WARNING1#2020-04-24T14:45:00
	WARNING1#2020-04-24T14:50:00

Para obtener solo los registros de advertencia de un dispositivo, la consulta se vuelve más específica con este esquema. La condición clave para la consulta utiliza la clave de partición `DeviceID="d#12345"` y la clave de clasificación `State#Date begins_with “WARNING”`. Esta consulta solo leerá los tres elementos relevantes con el estado de advertencia.

Paso 3: Abordar el patrón de acceso 4 (**`getLogsForOperatorBetweenTwoDates`**)

Puede importar DeviceStateLog_4.jsonD, donde se agregó el atributo `Operator` a la tabla `DeviceStateLog` con datos de ejemplo.

Primary key		Attributes		
Partition key: DeviceID	**Sort key: State#Date**			
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	State
		Liz	2020-04-24T14:55:00	NORMAL
	WARNING1#2020-04-24T14:40:00	Operator	Date	State
		Liz	2020-04-24T14:40:00	WARNING1
	WARNING1#2020-04-24T14:45:00	Operator	Date	State
		Liz	2020-04-24T14:45:00	WARNING1
	WARNING1#2020-04-24T14:50:00	Operator	Date	State
		Liz	2020-04-24T14:50:00	WARNING1
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	State
		Liz	2020-04-11T06:00:00	NORMAL
	NORMAL#2020-04-11T09:30:00	Operator	Date	State
		Sue	2020-04-11T09:30:00	NORMAL
	WARNING2#2020-04-11T09:25:00	Operator	Date	State
		Sue	2020-04-11T09:25:00	WARNING2
	WARNING3#2020-04-11T05:50:00	Operator	Date	State
		Sue	2020-04-11T05:50:00	WARNING3
	WARNING3#2020-04-11T05:55:00	Operator	Date	State
		Liz	2020-04-11T05:55:00	WARNING3
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	State
		Sue	2020-04-27T16:10:00	WARNING4
	WARNING4#2020-04-27T16:15:00	Operator	Date	State
		Sue	2020-04-27T16:15:00	WARNING4

Dado que `Operator` no es actualmente una clave de partición, no hay forma de realizar una búsqueda directa de clave-valor en esta tabla basada en `OperatorID`. Tendremos que crear una

nueva colección de elementos con un índice secundario global en `OperatorID`. El patrón de acceso requiere una búsqueda basada en fechas, por lo que Fecha es el atributo de clave de clasificación para el índice secundario global (GSI). Este es el aspecto actual del GSI:

Primary key		Attributes		
Partition key: Operator	**Sort key: Date**			
Liz	2020-04-11T05:55:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:55:00	WARNING3
	2020-04-11T06:00:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T06:00:00	NORMAL
	2020-04-24T14:40:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:40:00	WARNING1
	2020-04-24T14:45:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:45:00	WARNING1
	2020-04-24T14:50:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:50:00	WARNING1
	2020-04-24T14:55:00	DeviceID	State#Date	State
		d#12345	NORMAL#2020-04-24T14:55:00	NORMAL
Sue	2020-04-11T05:50:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:50:00	WARNING3
	2020-04-11T09:25:00	DeviceID	State#Date	State
		d#54321	WARNING2#2020-04-11T09:25:00	WARNING2
	2020-04-11T09:30:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T09:30:00	NORMAL
	2020-04-27T16:10:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:10:00	WARNING4
	2020-04-27T16:15:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:15:00	WARNING4

Para el patrón de acceso 4 (`getLogsForOperatorBetweenTwoDates`), puede consultar este GSI con clave de partición `OperatorID=Liz` y clave de clasificación `Date` entre `2020-04-11T05:58:00` y `2020-04-24T14:50:00`.

Primary key		Attributes		
Partition key: Operator	**Sort key: Date**			
	2020-04-11T05:55:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:55:00	WARNING3
Liz	2020-04-11T06:00:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T06:00:00	NORMAL
	2020-04-24T14:40:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:40:00	WARNING1
	2020-04-24T14:45:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:45:00	WARNING1
	2020-04-24T14:50:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:50:00	WARNING1
	2020-04-24T14:55:00	DeviceID	State#Date	State
		d#12345	NORMAL#2020-04-24T14:55:00	NORMAL
Sue	2020-04-11T05:50:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:50:00	WARNING3
	2020-04-11T09:25:00	DeviceID	State#Date	State
		d#54321	WARNING2#2020-04-11T09:25:00	WARNING2
	2020-04-11T09:30:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T09:30:00	NORMAL
	2020-04-27T16:10:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:10:00	WARNING4
	2020-04-27T16:15:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:15:00	WARNING4

Paso 4: Abordar patrones de acceso 5 (**`getEscalatedLogsForSupervisor`**), 6 (**`getEscalatedLogsWithSpecificStatusForSupervisor`**) y 7 (**`getEscalatedLogsWithSpecificStatusForSupervisorForDate`**)

Utilizaremos un índice disperso para abordar estos patrones de acceso.

Los índices secundarios globales son dispersos de forma predeterminada, por lo que solo aparecerán en el índice los elementos de la tabla base que contengan atributos de clave principal del índice. Esta es otra forma de excluir elementos que no son relevantes para el patrón de acceso que se está modelando.

Puede importar DeviceStateLog_6.json, donde se agregó el atributo `EscalatedTo` a la tabla `DeviceStateLog` con datos de ejemplo. Como ya se ha mencionado, no todos los registros se escalan a un supervisor.

Primary key		Attributes			
Partition key: DeviceID	**Sort key: State#Date**				
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	State	
		Liz	2020-04-24T14:55:00	NORMAL	
	WARNING1#2020-04-24T14:40:00	Operator	Date	State	
		Liz	2020-04-24T14:40:00	WARNING1	
	WARNING1#2020-04-24T14:45:00	Operator	Date	State	
		Liz	2020-04-24T14:45:00	WARNING1	
	WARNING1#2020-04-24T14:50:00	Operator	Date	State	
		Liz	2020-04-24T14:50:00	WARNING1	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	State	
		Liz	2020-04-11T06:00:00	NORMAL	
	NORMAL#2020-04-11T09:30:00	Operator	Date	State	
		Sue	2020-04-11T09:30:00	NORMAL	
	WARNING2#2020-04-11T09:25:00	Operator	Date	State	
		Sue	2020-04-11T09:25:00	WARNING2	
	WARNING3#2020-04-11T05:50:00	Operator	Date	State	
		Sue	2020-04-11T05:50:00	WARNING3	
	WARNING3#2020-04-11T05:55:00	Operator	Date	State	
		Liz	2020-04-11T05:55:00	WARNING3	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	State	
		Sue	2020-04-27T16:10:00	WARNING4	
	WARNING4#2020-04-27T16:15:00	Operator	Date	State	EscalatedTo
		Sue	2020-04-27T16:15:00	WARNING4	Sara

Ahora puede crear un nuevo GSI en el que `EscalatedTo` es la clave de partición y `State#Date` es la clave de clasificación. Observe que solo aparecen en el índice los elementos que tienen los atributos `EscalatedTo` y `State#Date`.

Primary key		Attributes			
Partition key: EscalatedTo	Sort key: State#Date				
		DeviceID	Operator	Date	State
Sara	WARNING4#2020-04-27T16:15:00	d#11223	Sue	2020-04-27T16:15:00	WARNING4

El resto de los patrones de acceso se resumen como sigue:

En la tabla siguiente se resumen todos los patrones de acceso y cómo los aborda el diseño del esquema:

Patrón de acceso	Tabla base/ GSI/LSI	Operación	Valor de clave de partición	Valor de la clave de clasificación	Otras condiciones/ filtros
createLogEntryForSpecificDevice	Tabla base	PutItem	DeviceID= deviceId	State#Date=state#date	
getLogsForSpecificDevice	Tabla base	Consultar	DeviceID= deviceId	State#Date begins_with “state1#”	ScanIndexForward = False
getWarningLogsForSpecificDevice	Tabla base	Consultar	DeviceID= deviceId	State#Date begins_with “WARNING”	
getLogsForOperatorBetweenTwoDates	GSI-1	Consultar	Operator= operatorName	Date between date1 and date2	
getEscalatedLogsForSupervisor	GSI-2	Consultar	EscalatedTo=supervisorName		

Patrón de acceso	Tabla base/ GSI/LSI	Operación	Valor de clave de partición	Valor de la clave de clasificación	Otras condiciones/ filtros
getEscalatedLogsWithSpecificStatusForSupervisor	GSI-2	Consultar	EscalatedTo=supervisorName	State#Date begins_with “state1#”	
getEscalatedLogsWithSpecificStatusForSupervisorForDate	GSI-2	Consultar	EscalatedTo=supervisorName	State#Date begins_with “state1#date1”	

Esquema final

Estos son los diseños finales del esquema. Para descargar este diseño de esquema como un archivo JSON, consulte los ejemplos de DynamoDB en GitHub.

Tabla base

Primary key		Attributes			
Partition key: DeviceID	**Sort key: State#Date**				
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date	State	
		Liz	2020-04-24T14:55:00	NORMAL	
	WARNING1#2020-04-24T14:40:00	Operator	Date	State	
		Liz	2020-04-24T14:40:00	WARNING1	
	WARNING1#2020-04-24T14:45:00	Operator	Date	State	
		Liz	2020-04-24T14:45:00	WARNING1	
	WARNING1#2020-04-24T14:50:00	Operator	Date	State	
		Liz	2020-04-24T14:50:00	WARNING1	
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date	State	
		Liz	2020-04-11T06:00:00	NORMAL	
	NORMAL#2020-04-11T09:30:00	Operator	Date	State	
		Sue	2020-04-11T09:30:00	NORMAL	
	WARNING2#2020-04-11T09:25:00	Operator	Date	State	
		Sue	2020-04-11T09:25:00	WARNING2	
	WARNING3#2020-04-11T05:50:00	Operator	Date	State	
		Sue	2020-04-11T05:50:00	WARNING3	
	WARNING3#2020-04-11T05:55:00	Operator	Date	State	
		Liz	2020-04-11T05:55:00	WARNING3	
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date	State	
		Sue	2020-04-27T16:10:00	WARNING4	
	WARNING4#2020-04-27T16:15:00	Operator	Date	State	EscalatedTo
		Sue	2020-04-27T16:15:00	WARNING4	Sara

GSI-1

Primary key		Attributes		
Partition key: Operator	**Sort key: Date**			
Liz	2020-04-11T05:55:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:55:00	WARNING3
	2020-04-11T06:00:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T06:00:00	NORMAL
	2020-04-24T14:40:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:40:00	WARNING1
	2020-04-24T14:45:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:45:00	WARNING1
	2020-04-24T14:50:00	DeviceID	State#Date	State
		d#12345	WARNING1#2020-04-24T14:50:00	WARNING1
	2020-04-24T14:55:00	DeviceID	State#Date	State
		d#12345	NORMAL#2020-04-24T14:55:00	NORMAL
Sue	2020-04-11T05:50:00	DeviceID	State#Date	State
		d#54321	WARNING3#2020-04-11T05:50:00	WARNING3
	2020-04-11T09:25:00	DeviceID	State#Date	State
		d#54321	WARNING2#2020-04-11T09:25:00	WARNING2
	2020-04-11T09:30:00	DeviceID	State#Date	State
		d#54321	NORMAL#2020-04-11T09:30:00	NORMAL
	2020-04-27T16:10:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:10:00	WARNING4
	2020-04-27T16:15:00	DeviceID	State#Date	State
		d#11223	WARNING4#2020-04-27T16:15:00	WARNING4

GSI-2

Primary key		Attributes			
Partition key: EscalatedTo	Sort key: State#Date				
Sara	WARNING4#2020-04-27T16:15:00	DeviceID	Operator	Date	State
		d#11223	Sue	2020-04-27T16:15:00	WARNING4

Uso de NoSQL Workbench con este diseño de esquema

Puede importar este esquema final en NoSQL Workbench, una herramienta visual que proporciona características de modelado de datos, visualización de datos y desarrollo de consultas para DynamoDB, a fin de explorar y editar más a fondo el nuevo proyecto. Para comenzar, siga estos pasos:

1. Descargue NoSQL Workbench. Para obtener más información, consulte the section called “Descargar”.
2. Descargue el archivo de esquema JSON que se muestra anteriormente, que ya está en el formato de modelo NoSQL Workbench.
3. Importe el archivo de esquema JSON en NoSQL Workbench. Para obtener más información, consulte the section called “Importación de un modelo existente”.
4. Una vez que haya importado en NOSQL Workbench, podrá editar el modelo de datos. Para obtener más información, consulte the section called “Edición de un modelo existente”.
5. Para visualizar el modelo de datos, agregar datos de ejemplo o importar datos de ejemplo de un archivo CSV, utilice la característica Visualizador de datos de NoSQL Workbench.

Uso de DynamoDB como almacén de datos para una tienda en línea

En este caso de uso se habla de utilizar DynamoDB como almacén de datos para una tienda en línea (o tienda electrónica).

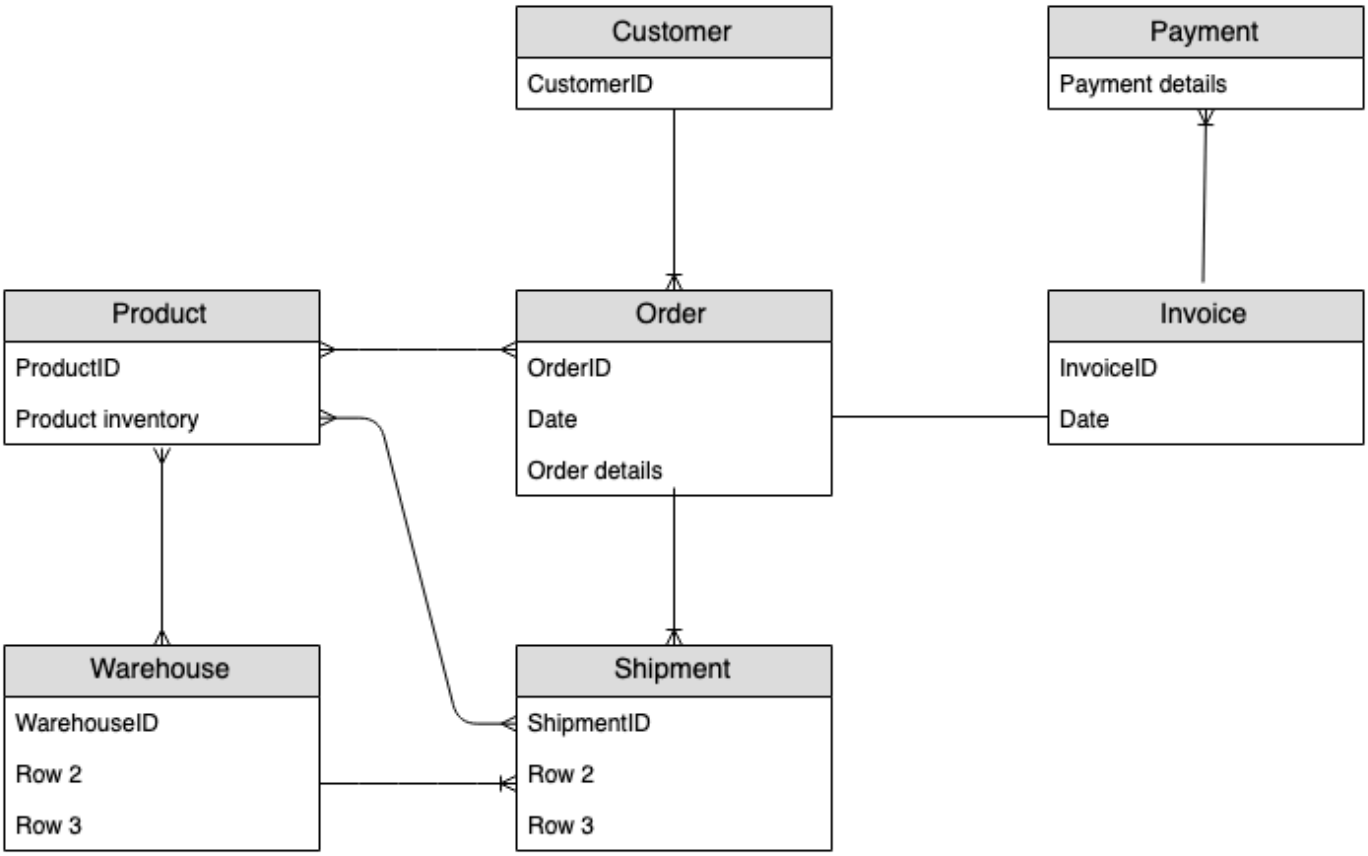
Caso de uso

Una tienda en línea permite a los usuarios navegar por diferentes productos y, finalmente, comprarlos. Basándose en la factura generada, el cliente puede pagar utilizando un código de descuento o una tarjeta regalo y, a continuación, abonar el importe restante con una tarjeta de crédito. Los productos adquiridos se retirarán de uno de varios almacenes y se enviarán a la dirección facilitada. Los patrones de acceso típicos de una tienda en línea incluyen:

- Obtener el cliente para un customerId dado
- Obtener el producto para un productId dado
- Obtener almacén para un warehouseId dado
- Obtener un inventario de productos para todos los almacenes por un productId
- Obtener pedido para un orderId dado
- Obtener todos los productos para un orderId dado
- Obtener la factura de un orderId dado
- Obtener todos los envíos para un orderId dado
- Obtener todos los pedidos de un productId determinado para un intervalo de fechas dado
- Obtener la factura para un invoiceId dado
- Obtener todos los pagos para un invoiceId dado
- Obtener los detalles de envío para un shipmentId dado
- Obtener todos los envíos para un warehouseId dado
- Obtener el inventario de todos los productos para un warehouseId dado
- Obtener todas las facturas de un customerId determinado para un intervalo de fechas dado
- Obtener todos los productos pedidos por un customerId determinado para un intervalo de fechas dado

Diagrama de relaciones entre entidades

Este es el diagrama de relaciones entre entidades (ERD) que utilizaremos para modelar DynamoDB como almacén de datos para una tienda en línea.



Patrones de acceso

Estos son los patrones de acceso que tendremos en cuenta al utilizar DynamoDB como almacén de datos para una tienda en línea.

1. `getCustomerByCustomerId`
2. `getProductByProductId`
3. `getWarehouseByWarehouseId`
4. `getProductInventoryByProductId`
5. `getOrderDetailsByOrderId`
6. `getProductByOrderId`
7. `getInvoiceByOrderId`
8. `getShipmentByOrderId`
9. `getOrderByProductIdForDateRange`
10. `getInvoiceByInvoiceId`

11getPaymentByInvoiceId

12getShipmentDetailsByShipmentId

13getShipmentByWarehouseId

14getProductInventoryByWarehouseId

15getInvoiceByCustomerIdForDateRange

16getProductsByCustomerIdForDateRange

Evolución del diseño de esquema

Mediante NoSQL Workbench para DynamoDB, importe AnOnlineShop_1.json para crear un nuevo modelo de datos llamado AnOnlineShop y una nueva tabla llamada OnlineShop. Tenga en cuenta que utilizamos los nombres genéricos PK y SK para la clave de partición y la clave de clasificación. Se trata de una práctica utilizada para almacenar distintos tipos de entidades en la misma tabla.

Paso 1: Abordar el patrón de acceso 1 (**getCustomerByCustomerId**)

Importe AnOnlineShop_2.json para gestionar el patrón de acceso 1 (getCustomerByCustomerId). Algunas entidades no tienen relaciones con otras entidades, por lo que utilizaremos el mismo valor de PK y SK para ellas. En los datos del ejemplo, observe que las claves utilizan un prefijo c# para distinguir customerId de otras entidades que se agregarán posteriormente. Esta práctica se repite también para otras entidades.

Para abordar este patrón de acceso, se puede utilizar una operación GetItem con PK=customerId y SK=customerId.

Paso 2: Abordar el patrón de acceso 2 (**getProductByProductId**)

Importe AnOnlineShop_3.json para abordar el patrón de acceso 2 (getProductByProductId) para la entidad product. Las entidades de producto tienen el prefijo p# y se ha utilizado el mismo atributo de clave de clasificación para almacenar customerID y productID. La denominación genérica y la partición vertical nos permiten crear tales colecciones de elementos para un diseño eficaz de tabla única.

Para abordar este patrón de acceso, se puede utilizar una operación GetItem con PK=productId y SK=productId.

Paso 3: Abordar el patrón de acceso 3 (**getWarehouseByWarehouseId**)

Importe AnOnlineShop_4.json para abordar el patrón de acceso 3 (`getWarehouseByWarehouseId`) para la entidad `warehouse`. Actualmente tenemos las entidades `customer`, `product` y `warehouse` agregadas a la misma tabla. Se distinguen mediante prefijos y el atributo `EntityType`. Un atributo de tipo (o prefijo de nomenclatura) mejora la legibilidad del modelo. La legibilidad se vería afectada si simplemente almacenáramos ID alfanuméricos para diferentes entidades en el mismo atributo. Sería difícil distinguir una entidad de otra en ausencia de estos identificadores.

Para abordar este patrón de acceso, se puede utilizar una operación `GetItem` con `PK=warehouseId` y `SK=warehouseId`.

Tabla base:

Primary key		Attributes		
Partition key: PK	Sort key: SK			
c#12345	c#12345	EntityType	Email	Name
		customer	samaneh@example.com	Samaneh Utter
p#12345	p#12345	EntityType	Detail	Price
		product	{"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}}	100
w#12345	w#12345	EntityType	Address	
		warehouse	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}}	

Paso 4: Abordar el patrón de acceso 4 (**`getProductInventoryByProductId`**)

Importe AnOnlineShop_5.json para abordar el patrón de acceso 4 (`getProductInventoryByProductId`). La entidad `warehouseItem` se utiliza para realizar un seguimiento del número de productos de cada almacén. Este elemento se actualizará normalmente cuando se agregue o elimine un producto de un almacén. Como se ve en el ERD, existe una relación de varios a varios entre `product` y `warehouse`. Aquí, la relación de uno a varios de `product` a `warehouse` se modela como `warehouseItem`. Más adelante, también se modelará la relación de uno a varios de `warehouse` a `product`.

El patrón de acceso 4 puede abordarse con una consulta sobre `PK=ProductId` y `SK begins_with “w#“`.

Para obtener más información sobre `begins_with()` y otras expresiones que pueden aplicarse a las claves de clasificación, consulte Expresiones de condición de clave.

Tabla base:

Primary key		Attributes		
Partition key: PK	Sort key: SK			
c#12345	c#12345	EntityType	Email	Name
		customer	samaneh@example.com	Samaneh Utter
	p#12345	EntityType	Detail	Price
		product	{"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}}	100
p#12345	w#12345	EntityType	Quantity	
		warehouseItem	50	
w#12345	w#12345	EntityType	Address	
		warehouse	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}}	

Paso 5: Abordar patrones de acceso 5 (**`getOrderDetailsByOrderId`**) y 6 (**`getProductByOrderId`**)

Agregue más elementos `customer`, `product` y `warehouse` a la tabla mediante la importación de AnOnlineShop_6.json. A continuación, importe AnOnlineShop_7.json para crear una colección de elementos a fin de que `order` pueda abordar los patrones de acceso 5 (`getOrderDetailsByOrderId`) y 6 (`getProductByOrderId`). Puede ver la relación de uno a varios entre `order` y `product` modelada como entidades orderItem.

Para abordar el patrón de acceso 5 (`getOrderDetailsByOrderId`), consulte la tabla con `PK=orderId`. Esto le proporcionará toda la información sobre el pedido, incluidos `customerId` y los productos solicitados.

Tabla base:

Primary key		Attributes		
Partition key: PK	Sort key: SK			
o#12345	c#12345	EntityType	Date	
		order	2020-06-21T19:10:00	
	p#12345	EntityType	Price	Quantity
		orderItem	100	2
	p#99887	EntityType	Price	Quantity
		orderItem	40	5

Para abordar el patrón de acceso 6 (`getProductByOrderId`), necesitamos leer los productos en un `order` solamente. Consulte la tabla con `PK=orderId` y `SK begins_with “p#”` para conseguirlo.

Tabla base:

Primary key		Attributes		
Partition key: PK	Sort key: SK			
	c#12345	EntityType	Date	
		order	2020-06-21T19:10:00	
o#12345	p#12345	EntityType	Price	Quantity
		orderItem	100	2
	p#99887	EntityType	Price	Quantity
		orderItem	40	5

Paso 6: Abordar el patrón de acceso 7 (**`getInvoiceByOrderId`**)

Importe AnOnlineShop_8.json para agregar una entidad `invoice` a la colección de elementos de pedido para gestionar el patrón de acceso 7 (`getInvoiceByOrderId`). Para abordar este patrón de acceso, puede utilizar una operación de consulta con `PK=orderId` y `SK begins_with “i#”`.

Tabla base:

Primary key		Attributes		
Partition key: PK	Sort key: SK			
o#12345	c#12345	EntityType	Date	
		order	2020-06-21T19:10:00	
	i#55443	EntityType	Amount	Date
		invoice	400	2020-06-21T19:18:00
	p#12345	EntityType	Price	Quantity
		orderItem	100	2
	p#99887	EntityType	Price	Quantity
		orderItem	40	5

Paso 7: Abordar el patrón de acceso 8 (**`getShipmentByOrderId`**)

Importe AnOnlineShop_9.json para agregar entidades `shipment` a la colección de elementos de pedido para abordar el patrón de acceso 8 (`getShipmentByOrderId`). Estamos ampliando el mismo modelo particionado verticalmente mediante la adición de más tipos de entidades en un diseño de tabla única. Observe cómo la colección de elementos de pedido contiene las distintas relaciones que una entidad `order` tiene con las entidades `shipment`, `orderItem` y `invoice`.

Para obtener envíos por `orderId`, puede realizar una operación de consulta con `PK=orderId` y `SK begins_with “sh#”`.

Tabla base:

Primary key		Attributes				
Partition key: PK	**Sort key: SK**					
o#12345	c#12345	EntityType	Date			
		order	2020-06-21T19:10:00			
	i#55443	EntityType	Amount	Date		
		invoice	400	2020-06-21T19:18:00		
	p#12345	EntityType	Price	Quantity		
		orderItem	100	2		
	p#99887	EntityType	Price	Quantity		
		orderItem	40	5		
	sh#88899	EntityType	Address	Type	Date	Warehouseld
		shipment	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00	w#12376
	sh#98765	EntityType	Address	Type	Date	Warehouseld
		shipment	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00	w#12345

Paso 8: Abordar el patrón de acceso 9 (**`getOrderByProductIdForDateRange`**)

Hemos creado una colección de elementos de pedido en el paso anterior. Este patrón de acceso tiene nuevas dimensiones de búsqueda (`ProductID` y `Date`) que le obligan a escanear toda la tabla y filtrar los registros relevantes para obtener los elementos buscados. Para abordar este patrón de acceso, necesitaremos crear un índice secundario global (GSI). Importe AnOnlineShop_10.json para crear una nueva colección de elementos mediante el GSI que permite recuperar datos `orderItem` de varias colecciones de elementos de pedido. Los datos tienen ahora `GSI1-PK` y `GSI1-SK`, que serán la clave de partición y la clave de clasificación de `GSI1`, respectivamente.

DynamoDB rellena automáticamente los elementos que contienen los atributos clave de un GSI desde la tabla al GSI. No es necesario realizar manualmente ninguna inserción adicional en el GSI.

Para abordar el patrón de acceso 9, realice una consulta de `GSI1` con `GSI1-PK=productId` y `GSI1SK between (date1, date2)`.

Tabla base:

Primary key		Attributes				
Partition key: PK	**Sort key: SK**					
o#12345	p#12345	EntityType	GSI1-PK	GSI1-SK	Price	Quantity
		orderItem	p#12345	2020-06-21T19:18:00	100	2
	p#99887	EntityType	GSI1-PK	GSI1-SK	Price	Quantity
		orderItem	p#99887	2020-06-21T19:20:00	40	5

GSI1:

Primary key		Attributes				
Partition key: GSI1-PK	**Sort key: GSI1-SK**					
p#12345	2020-06-21T19:18:00	PK	SK	EntityType	Quantity	Price
		o#12345	p#12345	orderItem	2	100
p#99887	2020-06-21T19:20:00	PK	SK	EntityType	Quantity	Price
		o#12345	p#99887	orderItem	5	40

Paso 9: Abordar patrones de acceso 10 (**`getInvoiceByInvoiceId`**) y 11 (**`getPaymentByInvoiceId`**)

Importe AnOnlineShop_11.json para abordar los patrones de acceso 10 (`getInvoiceByInvoiceId`) y 11 (`getPaymentByInvoiceId`), ambos relacionados con `invoice`. Aunque se trata de dos patrones de acceso diferentes, se realizan con la misma condición clave. `Payments` se definen como un atributo con el tipo de datos de asignación en la entidad `invoice`.

> **Note**
>
> `GSI1-PK` y `GSI1-SK` se sobrecargan para almacenar información sobre diferentes entidades, de forma que se puedan servir múltiples patrones de acceso desde el mismo GSI. Para obtener más información sobre la sobrecarga de GSI, consulte Sobrecarga de índices secundarios globales en DynamoDB.

Para abordar los patrones de acceso 10 y 11, realice una consulta de `GSI1` con `GSI1-PK=invoiceId` y `GSI1-SK=invoiceId`.

GSI1:

Primary key		Attributes							
Partition key: GSI1-PK	Sort key: GSI1-SK	PK	SK	EntityType	GSI2-PK	GSI2-SK	Detail	Amount	Date
i#55443	i#55443	o#12345	i#55443	invoice	c#12345	i#2020-06-21T19:18:00	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00

Paso 10: Abordar patrones de acceso 12 (**`getShipmentDetailsByShipmentId`**) y 13 (**`getShipmentByWarehouseId`**)

Importe AnOnlineShop_12.json para abordar los patrones de acceso 12 (`getShipmentDetailsByShipmentId`) y 13 (`getShipmentByWarehouseId`).

Observe que se agregan entidades `shipmentItem` a la colección de elementos de pedido en la tabla base para poder recuperar todos los detalles sobre un pedido en una sola operación de consulta.

Tabla base:

Primary key		Attributes							
Partition key: PK	Sort key: SK								
o#12345	sh#88899	EntityType	GSI1-PK	GSI1-SK	GSI2-PK	GSI2-SK	Address	Type	Date
		shipment	sh#88899	sh#88899	w#12376	sh#88899	{"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}}	Express	2020-06-22T08:20:00
	sh#98765	EntityType	GSI1-PK	GSI1-SK	GSI2-PK	GSI2-SK	Address	Type	Date
		shipment	sh#98765	sh#98765	w#12345	sh#98765	{"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}}	Express	2020-06-22T10:20:00
	shp#12345	EntityType	GSI1-PK	GSI1-SK	Quantity				
		shipmentItem	sh#98765	p#99887	3				
	shp#54321	EntityType	GSI1-PK	GSI1-SK	Quantity				
		shipmentItem	sh#88899	p#99887	2				
	shp#55555	EntityType	GSI1-PK	GSI1-SK	Quantity				
		shipmentItem	sh#98765	p#12345	2				

Las claves de partición y clasificación de `GSI1` ya se han utilizado para modelar una relación de uno a varios entre `shipment` y `shipmentItem`. Para abordar el patrón de acceso 12 (`getShipmentDetailsByShipmentId`), realice una consulta de `GSI1` con `GSI1-PK=shipmentId` y `GSI1-SK=shipmentId`.

GSI1:

Primary key		Attributes							
Partition key: GSI1-PK	Sort key: GSI1-SK								
	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#54321	shipmentItem	2				
sh#88899	sh#88899	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#88899	shipment	w#12376	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00
	p#12345	PK	SK	EntityType	Quantity				
		o#12345	shp#55555	shipmentItem	2				
	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#12345	shipmentItem	3				
sh#98765	sh#98765	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#98765	shipment	w#12345	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00

Tendremos que crear otro GSI (`GSI2`) para modelar la nueva relación de uno a varios entre `warehouse` y `shipment` para el patrón de acceso 13 (`getShipmentByWarehouseId`). Para

abordar este patrón de acceso, realice una consulta de `GSI2` con `GSI2-PK=warehouseId` y `GSI2-SK begins_with “sh#”`.

GSI2:

Primary key		Attributes							
Partition key: GSI2-PK	Sort key: GSI2-SK								
		PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
w#12376	sh#88899	o#12345	sh#88899	shipment	sh#88899	sh#88899	{"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}}	Express	2020-06-22T08:20:00
		PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
w#12345	sh#98765	o#12345	sh#98765	shipment	sh#98765	sh#98765	{"Country": {"S":"Sweden"},"County": {"S":"Vastra Gotaland"},"City": {"S":"Goteborg"},"Street": {"S":"Slanbarsvagen"},"Number": {"S":"34"},"ZipCode": {"S":"41787"}}	Express	2020-06-22T10:20:00

Paso 11: Abordar patrones de acceso 14 (**`getProductInventoryByWarehouseId`**), 15 (**`getInvoiceByCustomerIdForDateRange`**) y 16 (**`getProductsByCustomerIdForDateRange`**)

Importe AnOnlineShop_13.json para agregar los datos relacionados con el siguiente conjunto de patrones de acceso. Para abordar el patrón de acceso 14 (`getProductInventoryByWarehouseId`), realice una consulta de `GSI2` con `GSI2-PK=warehouseId` y `GSI2-SK begins_with “p#”`.

GSI2:

Primary key		Attributes							
Partition key: GSI2-PK	**Sort key: GSI2-SK**								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
c#12345	i#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	p#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	p#2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	

Para abordar el patrón de acceso 15 (`getInvoiceByCustomerIdForDateRange`), realice una consulta de `GSI2` con `GSI2-PK=customerId` y `GSI2-SK between (i#date1, i#date2)`.

GSI2:

Primary key		Attributes							
Partition key: GSI2-PK	**Sort key: GSI2-SK**								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
c#12345	i#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	p#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	p#2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	

Para abordar el patrón de acceso 16 (`getProductsByCustomerIdForDateRange`), realice una consulta de `GSI2` con `GSI2-PK=customerId` y `GSI2-SK between (p#date1, p#date2)`.

GSI2:

Primary key		Attributes							
Partition key: GSI2-PK	Sort key: GSI2-SK								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
c#12345	i#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	p#2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	p#2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	

Note

En NoSQL Workbench, las facetas representan los diferentes patrones de acceso a los datos de una aplicación para DynamoDB. Las facetas le permiten ver un subconjunto de datos de una tabla, sin tener que ver registros que no cumplen las restricciones de la faceta. Las facetas se consideran una herramienta visual de modelado de datos y no existen como un constructo utilizable en DynamoDB, ya que son puramente una ayuda para modelar patrones de acceso.

Importe AnOnlineShop_facets.json para ver las facetas de este caso de uso.

En la tabla siguiente se resumen todos los patrones de acceso y cómo los aborda el diseño del esquema:

Patrón de acceso	Tabla base/GSI/LSI	Operación	Valor de clave de partición	Valor de la clave de clasificación
getCustomerByCustomerId	Tabla de base	GetItem	PK=customerId	SK=customerId
getProductByProductId	Tabla de base	GetItem	PK=productId	SK=productId
getWarehouseByWarehouseId	Tabla de base	GetItem	PK=warehouseId	SK=warehouseId
getProductInventoryByProductId	Tabla base	Consultar	PK=productId	SK begins_with “w#”
getOrderDetailsByOrderId	Tabla base	Consultar	PK=orderId	
getProductByOrderId	Tabla base	Consultar	PK=orderId	SK begins_with “p#”
getInvoiceByOrderId	Tabla base	Consultar	PK=orderId	SK begins_with “i#”
getShipmentByOrderId	Tabla base	Consultar	PK=orderId	SK begins_with “sh#”
getOrderByProductIdForDateRange	GSI1	Consultar	PK=productId	SK between date1 and date2
getInvoiceByInvoiceId	GSI1	Consultar	PK=invoiceId	SK=invoiceId

Patrón de acceso	Tabla base/GSI/ LSI	Operación	Valor de clave de partición	Valor de la clave de clasificación
getPaymentByInvoiceId	GSI1	Consultar	PK=invoiceId	SK=invoiceId
getShipmentDetailsByShipmentId	GSI1	Consultar	PK=shipmentId	SK=shipmentId
getShipmentByWarehouseId	GSI2	Consultar	PK=warehouseId	SK begins_with “sh#”
getProductInventoryByWarehouseId	GSI2	Consultar	PK=warehouseId	SK begins_with “p#”
getInvoiceByCustomerIdForDateRange	GSI2	Consultar	PK=customerId	SK between i#date1 and i#date2
getProductsByCustomerIdForDateRange	GSI2	Consultar	PK=customerId	SK between p#date1 and p#date2

Esquema final de la tienda en línea

Estos son los diseños finales del esquema. Para descargar este diseño de esquema como un archivo JSON, consulte Patrones de diseño de DynamoDB en GitHub.

Tabla base

Primary key		Attributes			
Partition key: PK	**Sort key: SK**				
c#12345	c#12345	EntityType	Email	Name	
		customer	samaneh@example.com	Samaneh	
c#23456	c#23456	EntityType	Email	Name	
		customer	kathleen@example.com	Kathleen	
c#54321	c#54321	EntityType	Email	Name	
		customer	henrik@example.com	Henrik	
p#12345	p#12345	EntityType	Detail	Price	
		product	{"Name":{"S":"Options Open"},"Description":{"S":"The latest album"}}	100	
	w#12345	EntityType	GSI2-PK	GSI2-SK	Quantity
		warehouseItem	w#12345	p#12345	50
p#99887	p#99887	EntityType	Detail	Price	
		product	{"Name":{"S":"The Book"},"Description":{"S":"The best book ever"}}	40	
	w#12345	EntityType	GSI2-PK	GSI2-SK	Quantity
		warehouseItem	w#12345	p#99887	4
	w#12376	EntityType	Quantity		
		warehouseItem	4		
w#12345	w#12345	EntityType	Address		
		warehouse	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"MainStreet"},"Number":{"S":"20"},"ZipCode":{"S":"41111"}}		
w#12376	w#12376	EntityType	Address		
		warehouse	{"Country":{"S":"Sweden"},"County":		

GSI1

Primary key		Attributes							
Partition key: GSI1-PK	**Sort key: GSI1-SK**								
p#12345	2020-06-21T19:18:00	PK	SK	EntityType	GSI2-PK	GSI2-SK	Price	Quantity	
		o#12345	p#12345	orderItem	c#12345	2020-06-21T19:18:00	100	2	
p#99887	2020-06-21T19:20:00	PK	SK	EntityType	GSI2-PK	GSI2-SK	Price	Quantity	
		o#12345	p#99887	orderItem	c#12345	2020-06-21T19:20:00	40	5	
i#55443	i#55443	PK	SK	EntityType	GSI2-PK	GSI2-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	c#12345	2020-06-21T19:18:00	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
sh#88899	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#54321	shipmentItem	2				
	sh#88899	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#88899	shipment	w#12376	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00
sh#98765	p#12345	PK	SK	EntityType	Quantity				
		o#12345	shp#55555	shipmentItem	2				
	p#99887	PK	SK	EntityType	Quantity				
		o#12345	shp#12345	shipmentItem	3				
	sh#98765	PK	SK	EntityType	GSI2-PK	GSI2-SK	Address	Type	Date
		o#12345	sh#98765	shipment	w#12345	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00

GSI2

Primary key		Attributes							
Partition key: GSI2-PK	**Sort key: GSI2-SK**								
w#12345	p#12345	PK	SK	EntityType	Quantity				
		p#12345	w#12345	warehouseItem	50				
	p#99887	PK	SK	EntityType	Quantity				
		p#99887	w#12345	warehouseItem	4				
	sh#98765	PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
		o#12345	sh#98765	shipment	sh#98765	sh#98765	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T10:20:00
c#12345	2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#12345	orderItem	p#12345	2020-06-21T19:18:00	100	2	
	2020-06-21T19:18:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Detail	Amount	Date
		o#12345	i#55443	invoice	i#55443	i#55443	{"Payments":{"L":[{"M":{"Type":{"S":"GiftCard"},"Amount":{"N":"100"},"Data":{"S":"GiftCard data here..."}}},{"M":{"Type":{"S":"MasterCard"},"Amount":{"N":"300"},"Data":{"S":"Payment data here..."}}}]}}	400	2020-06-21T19:18:00
	2020-06-21T19:20:00	PK	SK	EntityType	GSI1-PK	GSI1-SK	Price	Quantity	
		o#12345	p#99887	orderItem	p#99887	2020-06-21T19:20:00	40	5	
w#12376	sh#88899	PK	SK	EntityType	GSI1-PK	GSI1-SK	Address	Type	Date
		o#12345	sh#88899	shipment	sh#88899	sh#88899	{"Country":{"S":"Sweden"},"County":{"S":"Vastra Gotaland"},"City":{"S":"Goteborg"},"Street":{"S":"Slanbarsvagen"},"Number":{"S":"34"},"ZipCode":{"S":"41787"}}	Express	2020-06-22T08:20:00

Uso de NoSQL Workbench con este diseño de esquema

Puede importar este esquema final en NoSQL Workbench, una herramienta visual que proporciona características de modelado de datos, visualización de datos y desarrollo de consultas para DynamoDB, a fin de explorar y editar más a fondo el nuevo proyecto. Para comenzar, siga estos pasos:

1. Descargue NoSQL Workbench. Para obtener más información, consulte the section called “Descargar”.
2. Descargue el archivo de esquema JSON que se muestra anteriormente, que ya está en el formato de modelo NoSQL Workbench.
3. Importe el archivo de esquema JSON en NoSQL Workbench. Para obtener más información, consulte the section called “Importación de un modelo existente”.
4. Una vez que haya importado en NOSQL Workbench, podrá editar el modelo de datos. Para obtener más información, consulte the section called “Edición de un modelo existente”.
5. Para visualizar el modelo de datos, agregar datos de ejemplo o importar datos de ejemplo de un archivo CSV, utilice la característica Visualizador de datos de NoSQL Workbench.

Prácticas recomendadas para modelar datos relacionales en DynamoDB

Esta sección proporciona las prácticas recomendadas para modelar datos relacionales en Amazon DynamoDB. En primer lugar, presentamos los conceptos tradicionales de modelado de datos. A continuación, describimos las ventajas de utilizar DynamoDB frente a los sistemas tradicionales de administración de bases de datos relacionales: cómo elimina la necesidad de realizar operaciones JOIN y reduce la sobrecarga.

A continuación explicamos cómo diseñar una tabla de DynamoDB que se escale de forma eficiente. Por último, ofrecemos un ejemplo de cómo modelar datos relacionales en DynamoDB.

Temas

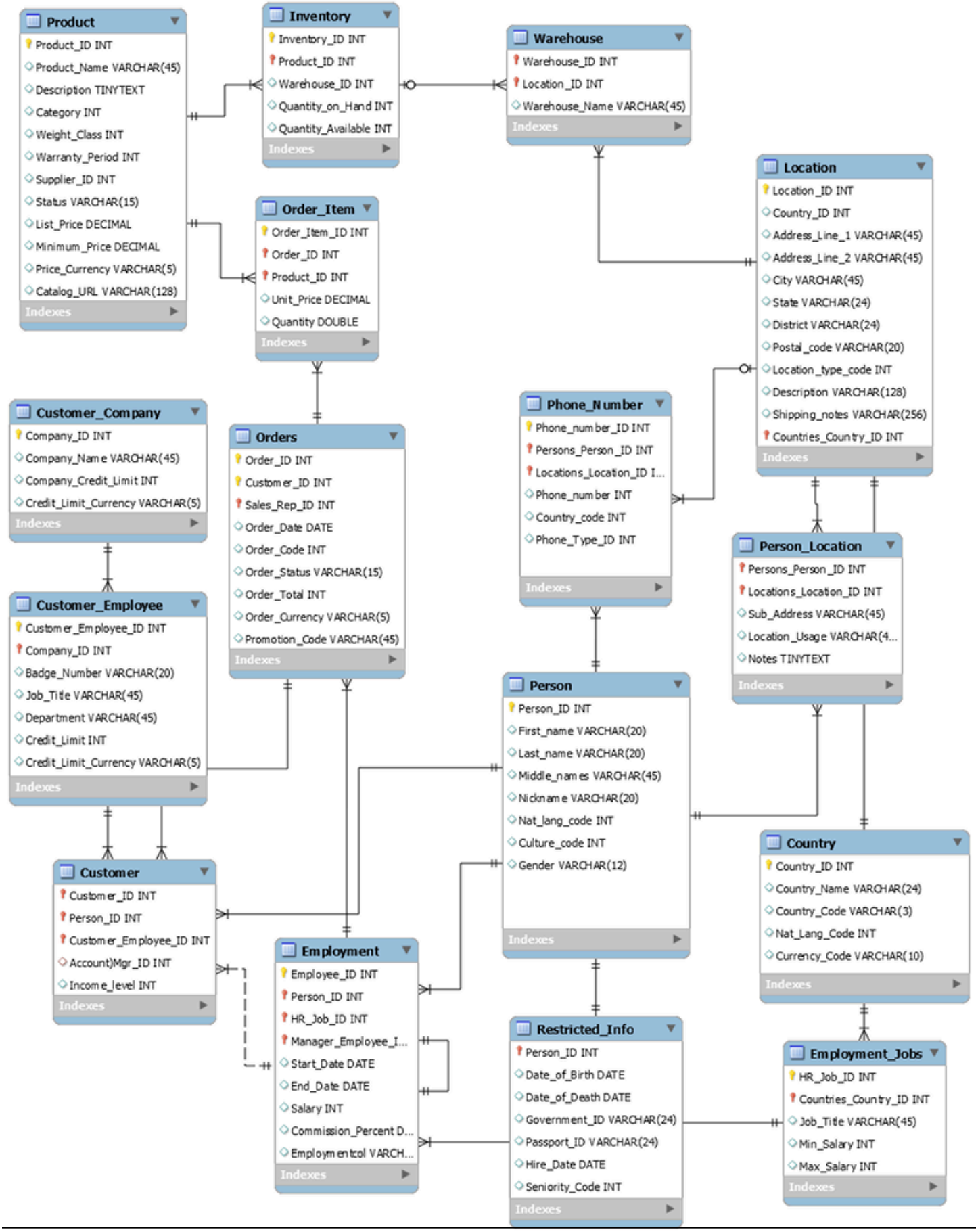
- Modelos tradicionales de bases de datos relacionales
- Cómo DynamoDB elimina la necesidad de las operaciones JOIN
- Cómo las transacciones de DynamoDB eliminan la sobrecarga del proceso de escritura
- Primeros pasos para modelar datos relacionales en DynamoDB

- Ejemplo de modelado de datos relacionales en DynamoDB

Modelos tradicionales de bases de datos relacionales

Un sistema tradicional de administración de bases de datos relacionales (RDBMS) almacena los datos en una estructura relacional normalizada. El objetivo del modelo de datos relacional es reducir la duplicación de datos (mediante la normalización) para respaldar la integridad referencial y reducir las anomalías en los datos.

El siguiente esquema es un ejemplo de modelo de datos relacional para una aplicación genérica de entrada de pedidos. La aplicación admite un esquema de recursos humanos que respalda los sistemas operativos y de apoyo empresarial de un fabricante teórico.


Product
Product_ID INT
Product_Name VARCHAR(45)
Description TINYTEXT
Category INT
Weight_Class INT
Warranty_Period INT
Supplier_ID INT
Status VARCHAR(15)
List_Price DECIMAL
Minimum_Price DECIMAL
Price_Currency VARCHAR(5)
Catalog_URL VARCHAR(128)
Indexes
Inventory
Inventory_ID INT
Product_ID INT
Warehouse_ID INT
Quantity_on_Hand INT
Quantity_Available INT
Indexes
Warehouse
Warehouse_ID INT
Location_ID INT
Warehouse_Name VARCHAR(45)
Indexes
Location
Location_ID INT
Country_ID INT
Address_Line_1 VARCHAR(45)
Address_Line_2 VARCHAR(45)
City VARCHAR(45)
State VARCHAR(24)
District VARCHAR(24)
Postal_code VARCHAR(20)
Location_type_code INT
Description VARCHAR(128)
Shipping_notes VARCHAR(256)
Countries_Country_ID INT
Indexes
Order_Item
Order_Item_ID INT
Order_ID INT
Product_ID INT
Unit_Price DECIMAL
Quantity DOUBLE
Indexes
Customer_Company
Company_ID INT
Company_Name VARCHAR(45)
Company_Credit_Limit INT
Credit_Limit_Currency VARCHAR(5)
Indexes
Orders
Order_ID INT
Customer_ID INT
Sales_Rep_ID INT
Order_Date DATE
Order_Code INT
Order_Status VARCHAR(15)
Order_Total INT
Order_Currency VARCHAR(5)
Promotion_Code VARCHAR(45)
Indexes
Phone_Number
Phone_number_ID INT
Persons_Person_ID INT
Locations_Location_ID I...
Phone_number INT
Country_code INT
Phone_Type_ID INT
Indexes
Person_Location
Persons_Person_ID INT
Locations_Location_ID INT
Sub_Address VARCHAR(45)
Location_Usage VARCHAR(4...
Notes TINYTEXT
Indexes
Customer_Employee
Customer_Employee_ID INT
Company_ID INT
Badge_Number VARCHAR(20)
Job_Title VARCHAR(45)
Department VARCHAR(45)
Credit_Limit INT
Credit_Limit_Currency VARCHAR(5)
Indexes
Person
Person_ID INT
First_name VARCHAR(20)
Last_name VARCHAR(20)
Middle_names VARCHAR(45)
Nickname VARCHAR(20)
Nat_lang_code INT
Culture_code INT
Gender VARCHAR(12)
Indexes
Country
Country_ID INT
Country_Name VARCHAR(24)
Country_Code VARCHAR(3)
Nat_Lang_Code INT
Currency_Code VARCHAR(10)
Indexes
Customer
Customer_ID INT
Person_ID INT
Customer_Employee_ID INT
Account)Mgr_ID INT
Income_level INT
Indexes
Employment
Employee_ID INT
Person_ID INT
HR_Job_ID INT
Manager_Employee_I...
Start_Date DATE
End_Date DATE
Salary INT
Commission_Percent D...
Employmentcol VARCH...
Indexes
Restricted_Info
Person_ID INT
Date_of_Birth DATE
Date_of_Death DATE
Government_ID VARCHAR(24)
Passport_ID VARCHAR(24)
Hire_Date DATE
Seniority_Code INT
Indexes
Employment_Jobs
HR_Job_ID INT
Countries_Country_ID INT
Job_Title VARCHAR(45)
Min_Salary INT
Max_Salary INT
Indexes

Como servicio de base de datos no relacional, DynamoDB ofrece muchas ventajas sobre los sistemas tradicionales de administración de bases de datos relacionales.

Cómo DynamoDB elimina la necesidad de las operaciones JOIN

Un sistema RDBMS utiliza un lenguaje de consulta estructurado (SQL) para devolver los datos a la aplicación. Debido a la normalización del modelo de datos, estas consultas suelen requerir la utilización del operador `JOIN` para combinar datos de una o varias tablas.

Por ejemplo, para generar una lista de pedidos de compra ordenados por la cantidad de existencias en todos los almacenes que puede enviar cada elemento, podría emitir la siguiente consulta SQL en el esquema anterior.

```
SELECT * FROM Orders
  INNER JOIN Order_Items ON Orders.Order_ID = Order_Items.Order_ID
  INNER JOIN Products ON Products.Product_ID = Order_Items.Product_ID
  INNER JOIN Inventories ON Products.Product_ID = Inventories.Product_ID
  ORDER BY Quantity_on_Hand DESC
```

Las consultas SQL de este tipo pueden proporcionar una API flexible para obtener acceso a los datos, pero requieren una gran cantidad de procesamiento. Cada unión de la consulta aumenta su complejidad en tiempo de ejecución, porque los datos de cada tabla deben prepararse y, a continuación, reunirse para devolver el conjunto de resultados.

Otros factores que pueden influir en el tiempo que tardan en ejecutarse las consultas son el tamaño de las tablas y si las columnas que se unen tienen índices. La consulta anterior inicia consultas complejas en varias tablas y, a continuación, ordena el conjunto de resultados.

En la esencia del modelado de datos NoSQL se encuentra la eliminación de la necesidad de `JOINs`. Ese es el motivo por el que creamos DynamoDB para que fuera compatible con Amazon.com y, por eso, DynamoDB puede ofrecer un rendimiento coherente a cualquier escala. Dada la complejidad en tiempo de ejecución de las consultas SQL y `JOINs`, el rendimiento del RBDMS no es constante a escala. Esto provoca problemas de rendimiento a medida que crecen las aplicaciones de los clientes.

Aunque la normalización de los datos reduce la cantidad de datos almacenados en el disco, a menudo los recursos más limitados que repercuten en el rendimiento son el tiempo de CPU y la latencia de red.

DynamoDB se creó para minimizar ambas restricciones mediante la eliminación de `JOINs` (y el fomento de la desnormalización de los datos) y la optimización de la arquitectura de base de

datos para responder completamente a la consulta de una aplicación con una única solicitud a un elemento. Estas cualidades permiten a DynamoDB ofrecer un rendimiento de un solo dígito en milisegundos a cualquier escala. Esto se debe a que la complejidad del tiempo de ejecución de las operaciones de DynamoDB es constante, independientemente del tamaño de los datos, para los patrones de acceso habituales.

Cómo las transacciones de DynamoDB eliminan la sobrecarga del proceso de escritura

Otro factor que puede ralentizar un sistema RDBMS es el uso de transacciones para escribir en un esquema normalizado. Como se muestra en el ejemplo, las estructuras de datos relacionales que se utilizan en la mayoría de aplicaciones de procesamiento de transacciones online (OLTP) deben dividirse y distribuirse por varias tablas lógicas cuando se almacenan en un sistema RDBMS.

Por tanto, se necesita un marco de transacciones compatible con ACID que evite las condiciones de carrera y los problemas de integridad de datos que podrían darse si una aplicación intentara leer un objeto que se estuviera escribiendo en ese momento. Un marco de transacciones de este tipo, cuando se combina con un esquema relacional, puede agregar una sobrecarga significativa al proceso de escritura.

La implementación de transacciones en DynamoDB prohíbe los problemas comunes de escalado que se encuentran con un RDBMS. DynamoDB lo hace emitiendo una transacción como una única llamada a la API y limitando el número de elementos a los que se puede acceder en esa única transacción. Las transacciones de larga duración pueden causar problemas operativos al mantener bloqueados los datos durante mucho tiempo o de forma perpetua, porque la transacción nunca se cierra.

Para evitar estos problemas en DynamoDB, las transacciones se implementaron con dos operaciones de la API distintas: `TransactWriteItems` y `TransactGetItems`. Estas operaciones de la API no tienen la semántica de inicio y fin que es común en un sistema RDBMS. Además, DynamoDB tiene un límite de acceso de 100 elementos en una transacción para evitar de forma similar las transacciones de larga duración. Para obtener más información sobre las transacciones de DynamoDB, consulte Uso de transacciones.

Por estos motivos, cuando su empresa requiere una respuesta de baja latencia ante consultas con mucho tráfico, suele aportar más ventaja el uso de un sistema NoSQL tanto desde un punto de vista técnico como económico. Amazon DynamoDB ayuda a resolver los problemas que limitan la escalabilidad del sistema relacional evitándolos.

El rendimiento de un sistema RDBMS no suele escalar bien por los siguientes motivos:

- Utiliza uniones caras para generar las vistas necesarias con los resultados de la consulta.
- Normalizan los datos y los guardan en varias tablas que requieren muchas consultas para escribir en el disco.
- Normalmente, hay que sumar los costos de desempeño de un sistema de transacciones compatible con ACID.

DynamoDB escala bien por los siguientes motivos:

- La flexibilidad del esquema permite que DynamoDB pueda almacenar datos jerárquicos complejos en un solo elemento.
- El diseño de la clave compuesta permite almacenar elementos relacionados próximos entre sí en la misma tabla.
- Las transacciones se realizan en una sola operación. El límite para el número de elementos a los que se puede acceder es de 100, para evitar operaciones de larga duración.

Las consultas que se realizan en el almacén de datos son mucho más sencillas y suelen tener la forma siguiente:

```
SELECT * FROM Table_X WHERE Attribute_Y = "somevalue"
```

DynamoDB trabaja mucho menos para devolver los datos solicitados que el sistema RDBMS del ejemplo anterior.

Primeros pasos para modelar datos relacionales en DynamoDB

> **Important**
>
> El diseño NoSQL requiere un modo de pensar distinto al diseño de RDBMS. En un sistema RDBMS, puede crear un modelo de datos normalizados sin pensar en los patrones de acceso. Posteriormente, podrá ampliar este modelo cuando surjan nuevos requisitos sobre preguntas y consultas. Por el contrario, en Amazon DynamoDB, no debe empezar a diseñar su esquema hasta que no sepa las preguntas que debe responder. Es absolutamente esencial conocer los problemas del negocio y los costos iniciales de los casos de uso de la aplicación.

Si desea comenzar a diseñar una tabla de DynamoDB que escale efectivamente, debe realizar primero varios pasos para identificar los patrones de acceso requeridos por los sistemas de ayuda de operaciones y de la actividad (OSS/BSS) que deben admitirse:

- En el caso de las nuevas aplicaciones, revise los casos de usuario para determinar las actividades y los objetivos. Documente los diferentes casos de uso que ha identificado y analice los patrones de acceso que requieren.
- En las aplicaciones existentes, analice los registros de consultas para saber cuántas personas utilizan actualmente el sistema y cuáles son los patrones de acceso de claves.

Tras completar este proceso, debería tener una lista similar a la siguiente:

	Most Common/Import Access Patterns in Our Organization
1	Look up employee details by employee ID
2	Query employee details by employee name
3	Find an employee's phone number(s)
4	Find a customer's phone number(s)
5	Get orders for a given customer within a given date range
6	Show all open orders within a given date range across all customers
7	See all employees hired recently
8	Find all employees working in a given warehouse
9	Get all items on order for a given product
10	Get current inventories for a given product at all warehouses
11	Get customers by account representative
12	Get orders by account representative and date
13	Get all employees with a given job title
14	Get inventory by product and warehouse
15	Get total product inventory
16	Get account representatives ranked by order total and sales period

En una aplicación real, la lista podría ser mucho más larga. Sin embargo, esta colección representa el rango de complejidad de los patrones de consulta que podría encontrar en un entorno de producción.

Un enfoque que suele utilizarse habitualmente para diseñar esquemas de DynamoDB es identificar las entidades de la capa de la aplicación y utilizar la desnormalización y la agregación de claves compuestas para reducir la complejidad de las consultas.

En DynamoDB, esto implica el uso de claves de ordenación complejas, índices secundarios globales sobrecargados, tablas o índices particionados y otros patrones de diseño. Puede utilizar estos elementos para estructurar los datos de forma que una aplicación pueda recuperar todo aquello que necesite para un determinado patrón de acceso realizando una sola consulta en una tabla o índice. El principal patrón que puede utilizar para modelar el esquema normalizado que se muestra en Modelos relacionales es el patrón de lista de adyacencia. Otros de los patrones que se pueden

utilizar en este diseño son la fragmentación de escrituras en índices secundarios globales, la sobrecarga de índices secundarios globales, las claves complejas y las agregaciones materializadas.

> **Important**
>
> Por lo general, en una aplicación de DynamoDB, es necesario mantener la menor cantidad de tablas posible. Puede haber excepciones, como los casos en los que hay implicados datos de serie temporal de gran volumen o conjuntos de datos que tienen diferentes patrones de acceso. Normalmente, basta una sola tabla con índices invertidos para permitir que, a través de consultas simples, se creen y recuperen las estructuras de datos jerárquicas y complejas que necesita la aplicación.

Para utilizar NoSQL Workbench para DynamoDB como ayuda para visualizar su diseño de clave de partición, consulte Creación de modelos de datos con NoSQL Workbench.

Ejemplo de modelado de datos relacionales en DynamoDB

En este ejemplo, se describe cómo se modelan los datos relacionales en Amazon DynamoDB. El diseño de las tablas de DynamoDB se corresponde con el esquema relacional de registro de pedidos que se describe en Modelos relacionales. Se ajusta al Patrón de diseño de listas de adyacencia, que es un mecanismo habitual para representar estructuras de datos relacionales en DynamoDB.

En este patrón de diseño, tiene que definir un conjunto de tipos de entidades, que normalmente tendrán correlación con las diferentes tablas del esquema relacional. Los elementos de entidad se agregan después a la tabla utilizando una clave principal compuesta (partición y ordenación). La clave de partición de estos elementos de entidad es el atributo que identifica el elemento de manera inequívoca y al que se hace referencia genéricamente en todos los elementos como PK. El atributo de la clave de ordenación contiene un valor que se puede usar como índice invertido o índice secundario global. Se identifica de forma genérica como SK.

Tiene que definir las siguientes entidades, que admiten el esquema relacional de registro de pedidos:

1. HR-Employee - PK: EmployeeID, SK: Employee Name
2. HR-Region - PK: RegionID, SK: Region Name
3. HR-Country - PK: CountryId, SK: Country Name
4. HR-Location - PK: LocationID, SK: Country Name
5. HR-Job - PK: JobID, SK: Job Title

6. HR-Department - PK: DepartmentID, SK: DepartmentName
7. OE-Customer - PK: CustomerID, SK: AccountRepID
8. OE-Order - PK OrderID, SK: CustomerID
9. OE-Product - PK: ProductID, SK: Product Name
10. OE-Warehouse - PK: WarehouseID, SK: Region Name

Después de agregar estos elementos de entidad a la tabla, puede definir sus relaciones agregando elementos de límite a las particiones de los elementos de entidad. En la tabla siguiente, se muestra este paso.
Tabla de ejemplo que contiene las relaciones entre los elementos de entidad.

En este ejemplo, las particiones `Employee`, `Order` y `Product Entity` de la tabla tienen elementos de límite adicionales que contienen marcadores que apuntan a otros elementos de entidad de la tabla. A continuación, debe definir algunos índices secundarios globales (GSI) que admitan todos los patrones de acceso definidos previamente. No todos los elementos de entidad usan el mismo tipo de valor para el atributo de la clave de ordenación o la clave principal. Todo lo que se necesita es que los atributos de la clave de ordenación y la clave principal estén presentes para insertarlos en la tabla.

El hecho de que algunas de estas entidades tengan nombres propios mientras que otras tienen identificadores de identidad como valores de la clave de ordenación permite que el mismo índice secundario global pueda admitir varios tipos de consultas. Esta técnica se denomina sobrecarga de GSI. Elimina de forma eficaz el límite predeterminado de 20 índices secundarios globales de las tablas que contienen varios tipos de elementos. Esto puede verse en el siguiente diagrama como GSI 1.
Tabla de ejemplo en la que se muestran índices secundarios globales que admiten varias consultas.

GSI 2 está diseñado para admitir un patrón de acceso de aplicaciones bastante común cuyo objetivo es obtener todos los elementos de la tabla que tienen un determinado estado. En el caso de las tablas grandes que tienen una distribución desigual de los elementos entre los estados disponibles, este patrón de acceso puede constituir un atajo, a menos que los elementos estén distribuidos entre varias particiones lógicas que puedan consultarse en paralelo. Este patrón de diseño se denomina `write sharding`.

Para realizar esta operación en GSI 2, la aplicación agrega el atributo de clave principal de GSI 2 a cada elemento Order. Lo rellena con un número aleatorio en el rango de 0 a N, donde N puede

calcularse genéricamente utilizando la siguiente fórmula, a menos que haya una razón específica para hacer lo contrario.

```
ItemsPerRCU = 4KB / AvgItemSize

PartitionMaxReadRate = 3K * ItemsPerRCU

N = MaxRequiredIO / PartitionMaxReadRate
```

Por ejemplo, supongamos que sus previsiones son las siguientes:

- Habrá hasta 2 millones de pedidos en el sistema, una cifra que aumentará hasta los 3 millones en 5 años.
- Hasta un 20 por ciento de estos pedidos tendrán el estado OPEN en un momento dado.
- El registro de pedido medio rondará los 100 bytes y habrá tres registros `OrderItem` en la partición de pedidos que tendrán 50 bytes cada uno, lo que hace un tamaño medio por entidad de pedido de 250 bytes.

En esa tabla, el cálculo del factor N sería similar al siguiente.

```
ItemsPerRCU = 4KB / 250B = 16

PartitionMaxReadRate = 3K * 16 = 48K

N = (0.2 * 3M) / 48K = 13
```

En este caso, tendrá que distribuir todos los pedidos entre al menos 13 particiones lógicas de GSI 2 para garantizar que la lectura de todos los elementos `Order` con el estado `OPEN` no genera una partición "caliente" en la capa de almacenamiento físico. Es conveniente dejar margen suficiente en este número para permitir anomalías en el conjunto de datos. Por tanto, es muy probable que un modelo con `N = 15` resulte adecuado. Como se mencionó anteriormente, lo hace agregando el valor aleatorio de 0 a N al atributo ISG 2 PK de cada registro `Order` y `OrderItem` que se inserta en la tabla.

En este desglose, se presupone que el patrón de acceso que tiene que recopilar todas las facturas OPEN tiene lugar con escasa frecuencia, de modo que puede utilizarse la capacidad de ráfagas para cumplimentar la solicitud. Puede consultar el siguiente índice secundario global utilizando una

condición de clave de ordenación `State` y `Date Range` para generar un subconjunto o todos los elementos `Orders` que tienen un determinado estado, según sea necesario.
Tabla de ejemplo en la que se muestra la clave principal de GSI 2 y los atributos proyectados.

En este ejemplo, los elementos se distribuyen aleatoriamente entre las 15 particiones lógicas. Esta estructura funciona porque el patrón de acceso requiere que se recuperen un gran número de elementos. Por tanto, es improbable que alguno de los 15 subprocesos devuelva conjuntos de resultados vacíos, lo que podría significar que hay capacidad que se está desaprovechando. Una consulta siempre utiliza una unidad de capacidad de lectura (RCU) o una unidad de capacidad de escritura (WCU), incluso si no se devuelve nada o no se escribe ningún dato.

Si el patrón de acceso necesita hacer una consulta de alta velocidad en este índice secundario global que devuelve un conjunto con escasos resultados, probablemente sea mejor usar un algoritmo hash para distribuir los elementos que un patrón aleatorio. En este caso, puede seleccionar un atributo que se conoce cuando la consulta se ejecuta en tiempo de ejecución y ese atributo se divide en un espacio de clave de 0 a 14 cuando se insertan los elementos. De ese modo, podría leerse de forma eficaz en el índice secundario global.

Por último, puede volver a consultar los patrones de acceso que se definieron anteriormente. A continuación, se muestra la lista con los patrones de acceso y las condiciones de consulta que utilizará con la nueva versión de DynamoDB de la aplicación para acomodarlos.

S. No.	Patrones de acceso	Condiciones de la consulta
1	Buscar detalles de empleado por ID de empleado	Clave principal en la tabla, ID="HR-EMPLOYEE"
2	Consultar detalles de empleado por nombre de empleado	Usar GSI-1, PK="Employee Name"
3	Obtener solo los detalles actuales del trabajo de un empleado	Clave principal en la tabla, PK=HR-EMPLOYEE-1, SK empieza por “JH”
4	Obtener pedidos de un cliente en un intervalo de fechas	Usar GSI-1, PK=CUSTOMER1, SK="STATUS-DATE", por cada StatusCode

S. No.	Patrones de acceso	Condiciones de la consulta
5	Mostrar todos los pedidos en estado OPEN en un intervalo de fechas en todos los clientes	Usar GSI-2, PK=consulta en paralelo para el intervalo [0..N], SK entre OPEN-Date1 y OPEN-Date2
6	Todos los empleados contratados recientemente	Usar GSI-1, PK="HR-CONFIDENTIAL", SK > date1
7	Buscar todos los empleados en un almacén específico	Usar GSI-1, PK=WAREHOUSE1
8	Obtener todos los artículos del pedido de un producto, incluidos los inventarios de la ubicación del almacén	Usar GSI-1, PK=PRODUCT1
9	Obtener clientes por representante de cuenta	Usar GSI-1, PK=ACCOUNT-REP
10	Obtener pedidos por representante de cuenta y fecha	Usar GSI-1, PK=ACCOUNT-REP, SK="STATUS-DATE", por cada StatusCode
11	Obtener todos los empleados con un cargo específico	Usar GSI-1, PK=JOBTITLE
12	Obtener inventario por producto y almacén	Clave principal en la tabla, PK=OE-PRODUCT1,SK=PRODUCT1
13	Obtener inventario total de productos	Clave principal en la tabla, PK=OE-PRODUCT1, SK=PRODUCT1

S. No.	Patrones de acceso	Condiciones de la consulta
14	Obtener los representantes de cuenta clasificados por el total de pedidos y el periodo de ventas	Usar GSI-1, PK=YYYY-Q1, scanIndexForward=False

Migración desde una base de datos relacional a DynamoDB

La migración de una base de datos relacional a DynamoDB requiere una planificación cuidadosa para garantizar un resultado satisfactorio. Esta guía lo ayudará a entender cómo funciona este proceso, qué herramientas tiene disponibles y a evaluar las posibles estrategias de migración para elegir la que mejor se adapte a sus necesidades.

Temas

- Razones para migrar a DynamoDB
- Consideraciones al migrar una base de datos relacional a DynamoDB
- Comprensión del funcionamiento de la migración a DynamoDB
- Herramientas que ayudan a migrar a DynamoDB
- Elección de la estrategia adecuada para migrar a DynamoDB
- Migración sin conexión a DynamoDB
- Migración híbrida a DynamoDB
- Migración en línea a DynamoDB mediante la migración de cada tabla de forma individual
- Migración en línea a DynamoDB con una tabla provisional personalizada

Razones para migrar a DynamoDB

La migración a Amazon DynamoDB presenta una serie de ventajas muy interesantes para las empresas y las organizaciones. Estas son algunas de las principales ventajas que convierten a DynamoDB en una opción atractiva para migrar las bases de datos:

- Escalabilidad: DynamoDB se ha diseñado para gestionar cargas de trabajo masivas y escalar sin complicaciones para adaptarse al crecimiento de los volúmenes de datos y del tráfico. DynamoDB le permite ampliar o reducir fácilmente su base de datos en función de la demanda, lo que garantiza que sus aplicaciones puedan gestionar picos repentinos de tráfico sin que afecte al rendimiento.
- Rendimiento: DynamoDB ofrece acceso a datos de baja latencia, lo que permite a las aplicaciones recuperar y procesar datos con una velocidad excepcional. Su arquitectura distribuida garantiza que las operaciones de lectura y escritura se distribuyan en varios nodos, lo que ofrece tiempos de respuesta uniformes en milisegundos de un solo dígito, incluso con altas tasas de solicitudes.

- Totalmente administrado: DynamoDB es un servicio totalmente administrado por AWS. Esto significa que AWS gestiona los aspectos operativos de la administración de bases de datos, incluidos el aprovisionamiento, la configuración, la aplicación de parches, las copias de seguridad y el escalado. Esto le permite centrarse más en el desarrollo de sus aplicaciones y menos en las tareas de administración de bases de datos.
- Arquitectura sin servidor: DynamoDB admite un modelo sin servidor, conocido como DynamoDB bajo demanda, en el que solo se paga por las solicitudes de lectura y escritura reales que realice la aplicación sin que sea necesario aprovisionar capacidad por adelantado. Este modelo de pago por solicitud ofrece rentabilidad y unos gastos operativos mínimos, ya que solo paga por los recursos que consume sin que sea necesario aprovisionar ni monitorear la capacidad.
- Flexibilidad NoSQL: a diferencia de las bases de datos relacionales tradicionales, DynamoDB sigue un modelo de datos NoSQL, lo que proporciona flexibilidad en el diseño de esquemas. DynamoDB permite almacenar datos estructurados, semiestructurados y sin estructurar, lo que los hace adecuados para gestionar tipos de datos diversos y en constante evolución. Esta flexibilidad permite ciclos de desarrollo más rápidos y una adaptación más sencilla a los requisitos empresariales en constante cambio.
- Alta disponibilidad y durabilidad: DynamoDB replica los datos en varias zonas de disponibilidad dentro de una región, lo que garantiza una alta disponibilidad y durabilidad de los datos. Además, gestiona automáticamente la replicación, la conmutación por error y la recuperación, lo que minimiza el riesgo de pérdida de datos o interrupciones en el servicio. DynamoDB ofrece un SLA de disponibilidad de hasta el 99,999 %.
- Seguridad y conformidad: DynamoDB se integra con AWS Identity and Access Management para controlar el acceso de forma precisa. Proporciona cifrado en reposo y en tránsito, lo que garantiza la seguridad de sus datos. DynamoDB también aplica varios estándares de conformidad, como HIPAA, PCI DSS y RGPD, lo que le permite cumplir con los requisitos normativos.
- Integración con el ecosistema de AWS: al formar parte del ecosistema de AWS, DynamoDB se integra perfectamente con otros servicios de AWS, como AWS Lambda, CloudFormation y AWS AppSync. Esta integración le permite crear arquitecturas sin servidor, aprovechar la infraestructura como código y crear aplicaciones basadas en datos en tiempo real.

Consideraciones al migrar una base de datos relacional a DynamoDB

Los sistemas de bases de datos relacionales y las bases de datos NoSQL tienen diferentes ventajas y desventajas. Estas diferencias hacen que el diseño de las bases de datos sea muy distinto entre los dos sistemas.

Tipo de tarea	Base de datos relacional	Base de datos NoSQL
Consultar la base de datos	En las bases de datos relacionales, los datos se pueden consultar de manera flexible, pero las consultas son relativamente caras y no escalan bien cuando hay mucho tráfico (consulte Primeros pasos para modelar datos relacionales en DynamoDB). Una aplicación de base de datos relacional puede implement ar la lógica empresarial en los procedimientos almacenad os, las subconsultas de SQL, las consultas de actualiza ción masiva y las consultas de agregación.	En una base de datos NoSQL como DynamoDB, existe un número limitado de métodos para consultar los datos; si se utiliza cualquier otro método diferente a estos, resultará más costoso y lento realizar las consultas. Las escrituras en DynamoDB son simples. La lógica empresarial de la aplicación que antes se ejecutaba en procedimientos almacenados debe refactori zarse para que se ejecute fuera de DynamoDB en un código personalizado que se ejecute en un host como Amazon EC2 o AWS Lambda.
Diseño de la base de datos	El diseño busca la flexibili dad sin preocuparse por los detalles o el rendimiento de la implementación. Por lo general, la optimización de consultas no afecta al diseño del esquema, pero la normalización es importante.	El esquema se diseña específicamente para que las consultas más habituales y más importantes resulten lo más rápidas y económicas que sea posible. Las estructur as de datos se ajustan a los requisitos específicos de los

Tipo de tarea	Base de datos relacional	Base de datos NoSQL
		casos de uso de la organización.

Diseñar para una base de datos NoSQL requiere un cambio de mentalidad respecto al diseño de un sistema de administración de bases de datos relacionales (RDBMS). En un sistema RDBMS, puede crear un modelo de datos normalizados sin pensar en los patrones de acceso. Posteriormente, podrá ampliar este modelo cuando surjan nuevos requisitos sobre preguntas y consultas. Puede organizar cada tipo de datos en su propia tabla.

Sin un diseño NoSQL, puede diseñar el esquema para DynamoDB cuando sepa las preguntas a las que tendrá que responder. Es esencial comprender los problemas empresariales y los patrones de lectura y escritura de la aplicación. También debería mantener el menor número de tablas posible en una aplicación de DynamoDB. Tener menos tablas hace que las cosas sean más escalables, requiere menos administración de permisos y reduce la sobrecarga de la aplicación de DynamoDB. También puede ayudar a mantener los costos de las copias de seguridad generalmente más bajos.

La tarea de modelar datos relacionales para DynamoDB y crear una nueva versión de la aplicación front-end se tratan en otro tema. En esta guía se da por sentado que tiene una nueva versión de la aplicación creada para usar DynamoDB, pero que aún debe determinar la mejor manera de migrar y sincronizar los datos históricos durante la transición.

Consideraciones sobre el tamaño

El tamaño máximo de cada elemento (fila) que se almacena en una tabla de DynamoDB es de 400 KB. Para obtener más información, consulte the section called “Cuotas”. El tamaño del elemento viene determinado por el tamaño total de todos los nombres y valores de los atributos de un elemento. Para obtener más información, consulte the section called “Tamaños y formatos de elementos”.

Si la aplicación necesita almacenar más datos en un elemento de lo que permite el límite de tamaño de DynamoDB, divida el elemento en una colección de elementos, comprima los datos del elemento o almacene el elemento como un objeto de Amazon Simple Storage Service (Amazon S3) y almacene el identificador de objetos de Amazon S3 en el elemento de DynamoDB. Consulte the section called “Elementos grandes”. El costo de actualizar un elemento se basa en el tamaño total del elemento. En el caso de las cargas de trabajo que requieren actualizaciones frecuentes de los elementos existentes, actualizar elementos pequeños de uno o dos KB costará menos que los

elementos más grandes. Para obtener más información sobre las colecciones de elementos, consulte the section called “Uso de colecciones de elementos”.

Al elegir los atributos de clave de clasificación y partición, otros ajustes de la tabla, el tamaño y la estructura de los elementos y si se van a crear índices secundarios, asegúrese de revisar la documentación de modelado de DynamoDB y la guía correspondiente para the section called “Optimización de costos”. Asegúrese de probar su plan de migración para que la solución de DynamoDB sea rentable y se ajuste a las características y limitaciones de DynamoDB.

Comprensión del funcionamiento de la migración a DynamoDB

Antes de revisar las herramientas de migración disponibles, piense en cómo procesa DynamoDB las escrituras.

La operación de escritura predeterminada y más común es una operación de API `PutItem` única. Puede realizar una operación `PutItem` en bucle para procesar conjuntos de datos. DynamoDB admite conexiones simultáneas prácticamente ilimitadas, por lo que, si puede configurar y ejecutar una rutina de carga masiva con varios subprocesos, como MapReduce o Spark, la velocidad de escritura solo está limitada por la capacidad de la tabla de destino (que también suele ser ilimitada).

Al cargar datos en DynamoDB, es importante comprender la velocidad de escritura del cargador. Si los elementos (filas) que está cargando tienen un tamaño de 1 KB o menos, esta velocidad es simplemente el número de elementos por segundo. A continuación, se puede aprovisionar la tabla de destino con suficientes WCU (unidades de capacidad de escritura) para gestionar esta velocidad. Si el cargador supera la capacidad aprovisionada por algún segundo, es posible que las solicitudes adicionales se limiten o se rechacen por completo. Puede comprobar si hay limitaciones en los gráficos de CloudWatch que se encuentran en la pestaña de monitoreo de la consola de DynamoDB.

La segunda operación que se puede realizar es con una API relacionada llamada `BatchWriteItem`. `BatchWriteItem` permite combinar hasta 25 solicitudes de escritura en una llamada a la API. El servicio las recibe y las procesa como solicitudes `PutItem` independientes en la tabla. Actualmente, al elegir `BatchWriteItem`, no disfrutará de las ventajas de los reintentos automáticos que se incluyen en el SDK de AWS para realizar llamadas individuales con `PutItem`. Por lo tanto, si hay algún error (por ejemplo, excepciones de limitación), tendrá que buscar en la lista de escrituras con errores en la llamada de respuesta a `BatchWriteItem`. Para obtener más información sobre cómo gestionar las advertencias de limitación en caso de que se detecten en los gráficos de limitación de CloudWatch, consulte the section called “Limitación”.

El tercer tipo de importación de datos es posible gracias a la característica Importación de DynamoDB desde S3. Esta característica le permite organizar un conjunto de datos de gran tamaño en Amazon S3 y solicitar a DynamoDB que importe automáticamente los datos a una tabla nueva. La importación no es instantánea y llevará un tiempo proporcional al tamaño del conjunto de datos. Sin embargo, es cómoda, ya que no requiere plataforma ETL ni código de DynamoDB personalizado. DynamoDB carga los datos en una tabla nueva que se crea mediante la importación. Actualmente, no le permite cargar datos en una tabla existente. DynamoDB importa los datos tal cual, sin transformaciones. Similar a `PutItem`, requiere un proceso previo y escribe los datos en el formato elegido en un bucket de Amazon S3.

Herramientas que ayudan a migrar a DynamoDB

Existen varias herramientas comunes de migración y ETL que puede utilizar para migrar datos a DynamoDB.

Amazon ofrece un host de herramientas de datos que se puede usar en migración, como AWS Database Migration Service (DMS), AWS Glue, Amazon EMR y Amazon Managed Streaming for Apache Kafka. Todas estas herramientas se pueden utilizar para realizar una migración durante el tiempo de inactividad y pueden aprovechar las características de la captura de datos de cambios (CDC) de la base de datos relacional para admitir las migraciones en línea. Al elegir una herramienta, será útil tener en cuenta el conjunto de destrezas y la experiencia que la organización tiene con cada herramienta, junto con las características, el rendimiento y el costo de cada una de ellas.

Muchos clientes optan por escribir sus propios scripts y trabajos de migración para crear transformaciones de datos personalizadas para el proceso de migración. Si tiene previsto operar una tabla de DynamoDB de gran volumen con mucho tráfico de escritura o con trabajos normales de gran carga masiva, puede que quiera escribir código de migración usted mismo para ganar confianza con el comportamiento de DynamoDB con un tráfico de escritura intenso. Al realizar una migración de práctica, se puede experimentar con situaciones como la gestión de la aceleración y el aprovisionamiento eficiente de tablas al principio del proyecto.

Elección de la estrategia adecuada para migrar a DynamoDB

Una aplicación de base de datos relacional grande puede abarcar cien o más tablas y admitir varias funciones de aplicación diferentes. Al realizar una migración grande, considere la posibilidad de dividir la aplicación en componentes o microservicios más pequeños y migrar un conjunto pequeño de tablas a la vez. A continuación, puede migrar componentes adicionales a DynamoDB en oleadas.

Al seleccionar una estrategia de migración, varios factores pueden orientarlo hacia una solución u otra. Podemos presentar estas opciones en un árbol de decisiones para simplificar las opciones disponibles en función de los requisitos y recursos disponibles. Los conceptos se mencionan brevemente aquí (pero se tratarán con más detalle más adelante en la guía):

- Migración sin conexión: si la aplicación puede tolerar algún tiempo de inactividad durante la migración, se simplificará el proceso de migración.
- Migración híbrida: este enfoque permite un tiempo de actividad parcial durante una migración, como permitir lecturas pero no escrituras, o permitir lecturas e inserciones, pero no actualizaciones ni eliminaciones.
- Migración en línea: las aplicaciones que no requieren ningún tiempo de inactividad durante la migración son más difíciles de migrar y pueden requerir una planificación significativa y un desarrollo personalizado. Una decisión clave es estimar y sopesar los costos de crear un proceso de migración personalizado frente al costo que supone para la empresa disponer de un periodo de inactividad durante la transición.

Si	Y	Entonces
Le parece bien cerrar la aplicación algún tiempo durante un periodo de mantenimiento para realizar la migración de datos. Esta migración se denomina migración sin conexión.		Utilice AWS DMS y realice una migración sin conexión con una tarea de carga completa. Si lo desea, dé forma previamente a los datos de origen con un SQL `VIEW`.
Le parece bien ejecutar la aplicación en modo de solo lectura durante la migración . Entonces la denominamos migración híbrida.		Desactive las escrituras en la aplicación o en la base de datos de origen. Utilice AWS DMS y realice una migración sin conexión con una tarea de carga completa.
Le parece bien ejecutar la aplicación con lecturas e inserciones de nuevos	Tiene conocimientos de desarrollo de aplicaciones y puede actualizar la aplicació	Utilice AWS DMS y realice una migración sin conexión con una tarea de carga

Si	Y	Entonces
registros, pero sin actualizarla ni eliminarla, durante la migración. Entonces la denominamos migración híbrida.	n relacional existente para realizar escrituras duales, incluso en DynamoDB, para todos los registros nuevos	completa. Al mismo tiempo, despliegue una versión de la aplicación existente que permita leer y realizar escrituras duales.
Necesita una migración con un tiempo de inactividad mínimo, Esta migración se denomina migración en línea.	• Está migrando las tablas de origen una por una a DynamoDB sin cambios importantes en el esquema.	Utilice AWS DMS para realizar una migración de datos en línea. Ejecute una tarea de carga masiva seguida de una tarea de sincronización de CDC.
Necesita una migración con un tiempo de inactividad mínimo, Esta migración se denomina migración en línea.	• Está combinando tablas de origen en menos tablas de DynamoDB según el esquema apilado o la filosofía de una sola tabla. • Tiene habilidades de desarrollo de bases de datos de backend y capacidad sobrante en el host de SQL.	Cree la tabla lista para NoSQL en la base de datos SQL. Rellénela y sincronícela con elementos JOIN, UNION, VIEW, desencadenadores y procedimientos almacenados.
Necesita una migración con un tiempo de inactividad mínimo, Esta migración se denomina migración en línea.	• Está combinando tablas de origen en menos tablas de DynamoDB según la filosofía de una sola tabla. Por ejemplo: • No tiene habilidades de desarrollo de bases de datos de backend ni capacidad sobrante en el host de SQL.	Tenga en cuenta los enfoques de migración híbrida o sin conexión.

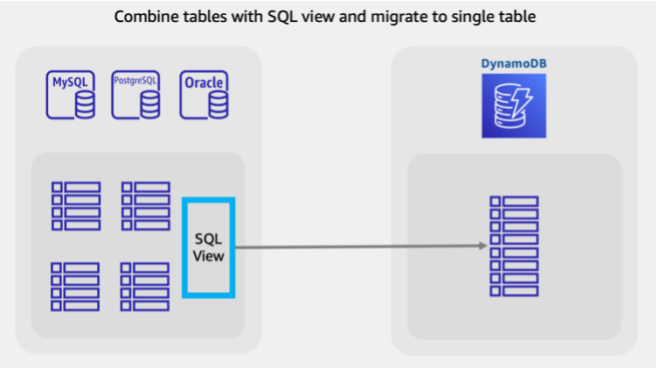
Si	Y	Entonces
Necesita una migración con un tiempo de inactividad mínimo, Esta migración se denomina migración en línea.	Le parece bien omitir la migración de los datos históricos de transacciones o puede archivarlos en Amazon S3 en lugar de migrarlos. Solo tiene que migrar unas cuantas tablas estáticas pequeñas.	Escriba un script o utilice cualquier herramienta ETL para migrar las tablas. Si lo desea, dé forma previamente a los datos de origen con un SQL `VIEW`.

Migración sin conexión a DynamoDB

Las migraciones sin conexión son adecuadas cuando se puede permitir un período de inactividad para realizar la migración. Las bases de datos relacionales suelen tener un tiempo de inactividad cada mes para realizar trabajos de mantenimiento y de aplicación de parches, o incluso tiempos de inactividad más largos en caso de actualizaciones de hardware o versiones importantes.

Amazon S3 se puede utilizar como área provisional durante una migración. Los datos almacenados en formato CSV (valores separados por comas) o JSON de DynamoDB se pueden importar automáticamente a una nueva tabla de DynamoDB con la característica Importación de DynamoDB desde S3.

Es posible que quiera combinar tablas para aprovechar patrones de acceso NoSQL únicos (por ejemplo, transformar cuatro tablas heredadas en una sola tabla de DynamoDB). Por lo general, una solicitud de documento con un único valor clave o una consulta de una recopilación de elementos agrupados previamente tiene una mejor latencia que una base de datos SQL que realiza una unión de varias tablas. Sin embargo, esto hace más difícil la tarea de migración. Una vista SQL podría hacer el trabajo dentro de la base de datos de origen para preparar un único conjunto de datos que represente las cuatro tablas de un conjunto.



Esta vista puede crear tablas `JOIN` en una forma desnormalizada o podría mantener las entidades normalizadas y apilar las tablas mediante un SQL `UNION`. En este vídeo se describen las decisiones clave en torno a la remodelación de los datos relacionales. Para las migraciones sin conexión, el uso de una vista para combinar tablas es una manera excelente de dar forma a los datos para un esquema de tabla única de DynamoDB.

Plan

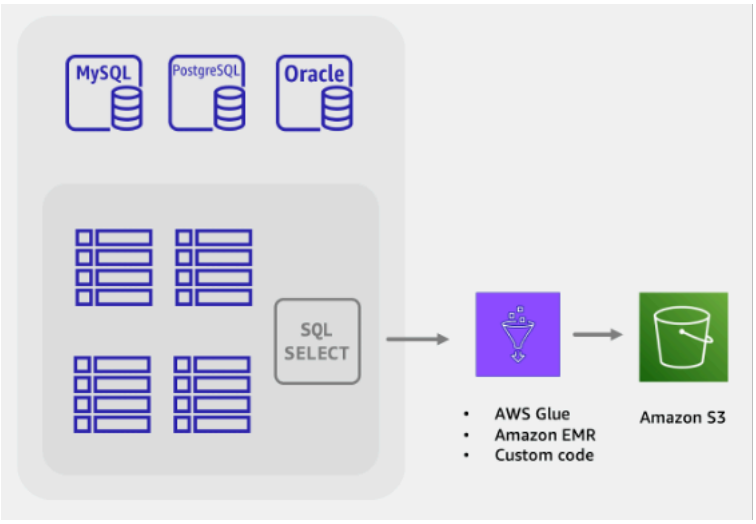
Realización de una migración sin conexión con Amazon S3

Herramientas

- Un trabajo ETL para extraer y transformar datos SQL y almacenarlos en un bucket de S3, como:
  - AWS Database Migration Service, un servicio que puede cargar datos históricos de forma masiva y también puede procesar registros de captura de datos de cambio (CDC) para sincronizar las tablas de origen y destino.
  - AWS Glue
  - Amazon EMR
  - Su propio código personalizado
- Característica Importación de DynamoDB desde S3

Pasos de la migración sin conexión:

1. Cree un trabajo ETL que pueda consultar la base de datos SQL, transformar los datos de las tablas en formato JSON o CSV de DynamoDB y guardarlos en un bucket de S3.



2. La característica Importación de DynamoDB desde S3 se invoca para crear una tabla nueva y cargar automáticamente los datos desde el bucket de S3.

La migración total sin conexión es sencilla y directa, pero puede que no sea muy popular entre los propietarios y los usuarios de las aplicaciones. Los usuarios se beneficiarían si la aplicación pudiera ofrecer niveles de servicio reducidos durante la migración, en lugar de no ofrecer ningún servicio.

Podría añadir una funcionalidad para desactivar las escrituras durante la migración sin conexión y, al mismo tiempo, permitir que las lecturas continuaran con normalidad. Los usuarios de la aplicación podrían seguir navegando y consultando los datos existentes de forma segura mientras se migran los datos relacionales. Si esto es lo que busca, siga leyendo para obtener más información sobre las migraciones híbridas.

Migración híbrida a DynamoDB

Si bien todas las aplicaciones de bases de datos realizan operaciones de lectura y escritura, se deben tener en cuenta los tipos de operaciones de escritura que se realizan al planificar una migración híbrida o en línea. Las escrituras en bases de datos se pueden clasificar en tres grupos: inserciones, actualizaciones y eliminaciones. Es posible que algunas aplicaciones no requieran el procesamiento inmediato de las eliminaciones. Estas aplicaciones pueden aplazar eliminaciones en un proceso de limpieza masiva a final de mes, por ejemplo. Estos tipos de aplicaciones pueden ser más fáciles de migrar y, al mismo tiempo, requieren un tiempo de actividad parcial.

Plan

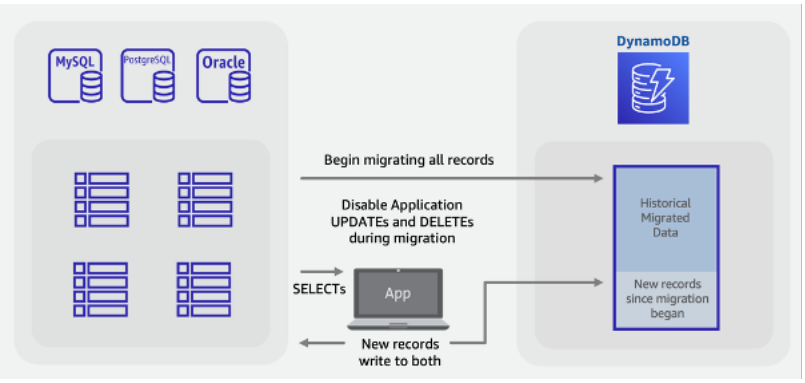
Realizar una migración híbrida en línea y sin conexión con escrituras duales de aplicaciones

Herramientas

- Un trabajo ETL para extraer y transformar datos SQL y almacenarlos en un bucket de S3, como:
  - AWS DMS
  - AWS Glue
  - Amazon EMR
  - Su propio código personalizado

Pasos de la migración híbrida:

1. Cree una tabla de DynamoDB de destino. Esta tabla recibirá tanto datos masivos históricos como datos nuevos y activos.
2. Cree una versión de la aplicación heredada que tenga desactivadas las eliminaciones y actualizaciones y, al mismo tiempo, realice todas las inserciones como escrituras duales tanto en la base de datos SQL como en DynamoDB.
3. Comience el trabajo ETL o la tarea de AWS DMS para reponer los datos existentes e implementar la nueva versión de la aplicación al mismo tiempo.
4. Cuando se complete el trabajo de reposición, DynamoDB dispondrá de todos los registros nuevos y existentes y estará listo para la transición de la aplicación.



> **Note**
>
> El trabajo de reposición escribe directamente desde SQL a DynamoDB. No podemos utilizar la característica de importación de S3 como en el ejemplo de migración sin conexión, ya que esa característica crea una tabla nueva que no se ejecutará en un entorno real hasta después de que DynamoDB cargue los datos.

Migración en línea a DynamoDB mediante la migración de cada tabla de forma individual

Muchas bases de datos relacionales tienen una característica llamada Captura de datos de cambios (CDC), que permite a los usuarios solicitar una lista de los cambios realizados en una tabla y que se han realizado antes o después de un momento específico. La CDC utiliza registros internos para activar esta característica y no requiere que la tabla tenga ninguna columna con marca temporal para que funcione.

Al migrar un esquema de tablas SQL a una base de datos NoSQL, es posible que desee combinar y cambiar la forma de los datos en menos tablas. Esto le permitirá recopilar datos en un solo lugar y evitar tener que unir manualmente los datos relacionados en operaciones de lectura de varios pasos. Sin embargo, no siempre es necesario dar forma a los datos de una sola tabla y, a veces, las tablas se migran una por una a DynamoDB. Estas migraciones de tablas individuales una por una son menos complicadas, ya que se puede aprovechar la característica CDC de la base de datos de origen con las herramientas ETL habituales que admiten este tipo de migración. Es posible que los datos de cada fila se sigan transformando en nuevos formatos, pero el alcance de cada tabla sigue siendo el mismo.

Considere la posibilidad de migrar tablas SQL una por una a DynamoDB, con la salvedad de que DynamoDB no admite uniones del servidor. Tendrá que agregar lógica a la aplicación para combinar los datos de varias tablas.

Plan

Realizar una migración en línea de cada tabla a DynamoDB con AWS DMS

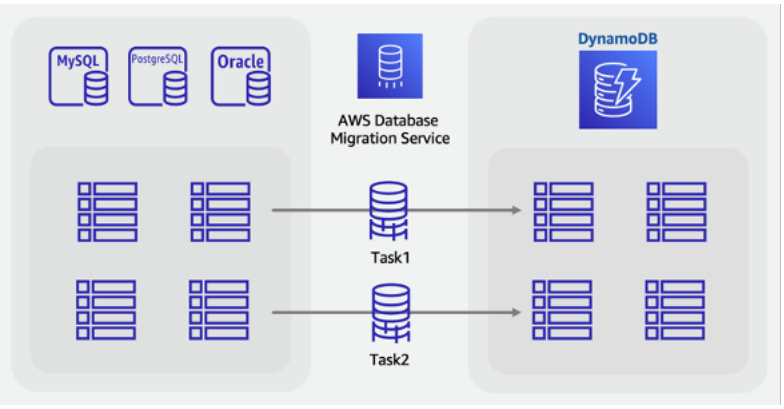
Herramientas

- AWS DMS (DMS)

Pasos de la migración en línea:

1. Identifique las tablas del esquema de origen que se van a migrar.
2. Cree el mismo número de tablas en DynamoDB con la misma estructura de claves que en el origen.
3. Cree un servidor de replicación en AWS DMS y configure los puntos de conexión de origen y destino.

4. Defina cualquier transformación por fila que sea necesaria (como columnas concatenadas o la conversión de fechas al formato de cadena ISO-8601).
5. Cree una tarea de migración para cada tabla para la carga completa y la captura de datos de cambios.
6. Supervise estas tareas hasta que comience la fase de replicación en curso.
7. Llegados a este punto, puede realizar cualquier auditoría de validación y, a continuación, cambiar los usuarios a la aplicación que lee y escribe en DynamoDB.



Migración en línea a DynamoDB con una tabla provisional personalizada

Como en el escenario de migración sin conexión anterior, tiene la opción de elegir combinar tablas para aprovechar patrones de acceso NoSQL únicos (por ejemplo, transformar cuatro tablas heredadas en una sola tabla de DynamoDB). Un SQL `VIEW` podría hacer el trabajo dentro de la base de datos de origen para preparar un único conjunto de datos que represente las cuatro tablas de un conjunto.

Sin embargo, para las migraciones en línea con datos cambiantes y activos, no podrá aprovechar las características de CDC, ya que no se admiten para `VIEW`. Si sus tablas incluyen una columna de marca de tiempo actualizada por última vez y estas se incorporan a la `VIEW`, puede crear un trabajo ETL personalizado que las utilice para lograr una carga masiva con la sincronización.

Un enfoque novedoso para este desafío es utilizar características SQL estándar, como vistas, procedimientos almacenados y desencadenadores para crear una nueva tabla SQL con el formato final NoSQL de DynamoDB deseado.

Si el servidor de base de datos tiene la capacidad sobrante, es posible crear esta tabla provisional única antes de que comience la migración. Haría esto mediante la escritura de un procedimiento almacenado que lea las tablas existentes, transforme los datos según sea necesario y escriba en la nueva tabla provisional. Puede agregar un conjunto de desencadenadores para replicar los cambios de las tablas en la tabla provisional en tiempo real. Si la política de la empresa no admite los desencadenadores, los cambios en los procedimientos almacenados pueden arrojar el mismo resultado. Usted añadiría unas cuantas líneas de código a cualquier procedimiento que escriba datos para, además, escribir los mismos cambios en la tabla provisional.

Disponer de esta tabla provisional, totalmente sincronizada con las tablas de aplicaciones heredadas, le proporcionará un excelente punto de partida para una migración en directo. Las herramientas que utilizan la CDC de base de datos para realizar migraciones en tiempo real, como AWS DMS, ahora se pueden utilizar en esta tabla. Una ventaja de este enfoque es que utiliza habilidades y características de SQL muy conocidas disponibles en el motor de bases de datos relacionales.

Plan

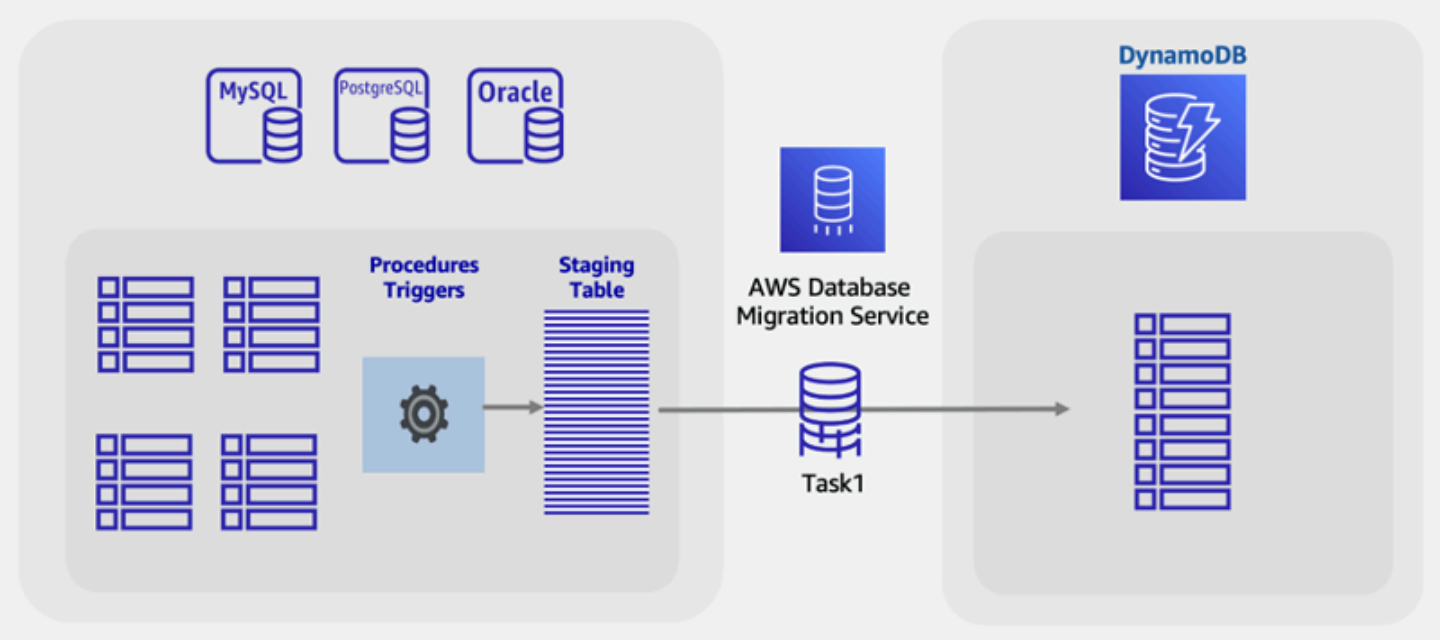
Realizar una migración en línea con una tabla provisional de SQL mediante AWS DMS

Herramientas

- Procedimientos o desencadenadores de SQL almacenados y personalizados
- AWS DMS

Pasos de la migración en línea:

1. En el motor de base de datos relacional de origen, asegúrese de que haya espacio en disco y capacidad de procesamiento adicionales.
2. Cree una nueva tabla provisional en la base de datos SQL, con las marcas de tiempo o las características CDC activadas.
3. Escriba y ejecute un procedimiento almacenado para copiar los datos de la tabla relacional existente en la tabla provisional.
4. Implemente desencadenadores o modifique los procedimientos existentes para realizar una escritura doble en la nueva tabla provisional y, al mismo tiempo, realizar escrituras normales en las tablas existentes.
5. Ejecute AWS DMS para migrar y sincronizar esta tabla de origen con una tabla de DynamoDB de destino.



Esta guía presenta varias consideraciones y enfoques para migrar datos de bases de datos relacionales a DynamoDB, centrándose en minimizar el tiempo de inactividad y utilizar herramientas y técnicas de bases de datos comunes. Para obtener más información, consulte los siguientes temas:

- AWS DMS Guía del usuario de
- AWS Glue Guía del usuario de
- Prácticas recomendadas para migrar desde RDBMS a DynamoDB

NoSQL Workbench para DynamoDB

NoSQL Workbench para Amazon DynamoDB es una aplicación GUI de cliente multiplataforma que puede usar para el desarrollo moderno de bases de datos y operaciones. Está disponible para Windows, macOS y Linux. NoSQL Workbench es una herramienta de desarrollo visual que proporciona características de modelado de datos, visualización de datos y desarrollo de consultas para ayudarle a diseñar, crear, consultar y administrar tablas de DynamoDB. NoSQL Workbench ahora incluye DynamoDB local como parte opcional del proceso de instalación, lo que facilita el modelado de los datos en DynamoDB local. Para obtener más información sobre DynamoDB local y sus requisitos, consulte Configuración de la versión de DynamoDB local (versión descargable) .

Modelado de datos

Con NoSQL Workbench para DynamoDB, puede crear nuevos modelos de datos o diseñar modelos basados en modelos de datos existentes que satisfagan los patrones de acceso a datos de su aplicación. También puede importar y exportar el modelo de datos diseñado al final del proceso. Para obtener más información, consulte Creación de modelos de datos con NoSQL Workbench.

Visualización de datos

El visualizador de modelos de datos proporciona un lienzo donde puede mapear las consultas y visualizar los patrones de acceso (facetas) de la aplicación sin tener que escribir código. Cada faceta corresponde a un patrón de acceso diferente en DynamoDB. Puede generar automáticamente datos de muestra para utilizarlos en su modelo de datos. Para obtener más información, consulte Visualización de patrones de acceso a datos.

Generación de operaciones

NoSQL Workbench proporciona una completa interfaz gráfica de usuario rica para que desarrolle y pruebe las consultas. Puede utilizar el generador de operaciones para visualizar, explorar y consultar conjuntos de datos en directo. El generador de operaciones estructuradas admite la expresión de proyecciones y de condiciones y, además, genera código de muestra en varios idiomas. Puede clonar tablas directamente de una cuenta de Amazon DynamoDB en otra en distintas regiones. También puede clonar tablas directamente entre DynamoDB local y una cuenta de Amazon DynamoDB para copiar más rápidamente el esquema de claves de la tabla (y, opcionalmente, el esquema y los elementos de GSI) entre sus entornos de desarrollo. Para obtener más información, consulte Exploración de conjuntos de datos y creación de operaciones con NoSQL Workbench.

En el siguiente vídeo, se detallan los conceptos del modelado de datos con NoSQL Workbench.

Temas

- Descargar NoSQL Workbench para DynamoDB
- Instalar NoSQL Workbench para DynamoDB
- Creación de modelos de datos con NoSQL Workbench
- Visualización de patrones de acceso a datos
- Exploración de conjuntos de datos y creación de operaciones con NoSQL Workbench
- Modelos de datos de ejemplo para NoSQL Workbench
- Historial de versiones de NoSQL Workbench

Descargar NoSQL Workbench para DynamoDB

Siga estas instrucciones para descargar NoSQL Workbench y DynamoDB local* para Amazon DynamoDB.

Requisitos previos

Las instalaciones de Ubuntu requieren dos componentes de software: libfuse2 y curl.

libfuse2

A partir de Ubuntu 22.04, libfuse2 ya no se instala de forma predeterminada. Para solucionar este problema, ejecute `sudo add-apt-repository universe && sudo apt install libfuse2` para instalar la versión más reciente de Ubuntu.

curl

Actualice Ubuntu, ejecute `sudo apt update && sudo apt upgrade`.

A continuación, instale cURL y ejecute `sudo apt install curl`.

Para descargar NoSQL Workbench y DynamoDB local

1. Descargue la versión apropiada de NoSQL Workbench para su sistema operativo.

Sistema operativo	Enlace de descarga
macOS (Intel)**	Descargar para macOS (Intel)
macOS (Apple Silicon)	Descargar para macOS (Apple Silicon)
Windows	Descargar para Windows
Linux***	Descargar para Linux

* NoSQL Workbench incluye DynamoDB local como parte opcional del proceso de instalación.

** Si aparece un mensaje de advertencia al intentar abrir NoSQL Workbench que indica que ningún desarrollador identificado ha registrado la aplicación en Apple, haga lo siguiente:

1. Localice la aplicación y ábrala.
2. Mantenga presionada la tecla Control y haga clic en el icono de la aplicación. A continuación, seleccione Abrir en el menú contextual.

   Al hacerlo, la aplicación se guarda como excepción en los ajustes de seguridad. Abra la aplicación haciendo doble clic en ella del mismo modo que con cualquier aplicación registrada.

*** NoSQL Workbench es compatible con Ubuntu 12.04, Fedora 21 y Debian 8 o cualquier versión más reciente de estas distribuciones de Linux.

2. Inicie la aplicación que ha descargado y, a continuación, siga los pasos que se muestran en Install NoSQL Workbench (Instalar NoSQL Workbench).

> **Note**
>
> Para ejecutar DynamoDB local se requiere una versión 11.x o más reciente del Entorno de ejecución de Java (JRE).

Instalar NoSQL Workbench para DynamoDB

Siga estos pasos para instalar NoSQL Workbench y DynamoDB local en una plataforma compatible.

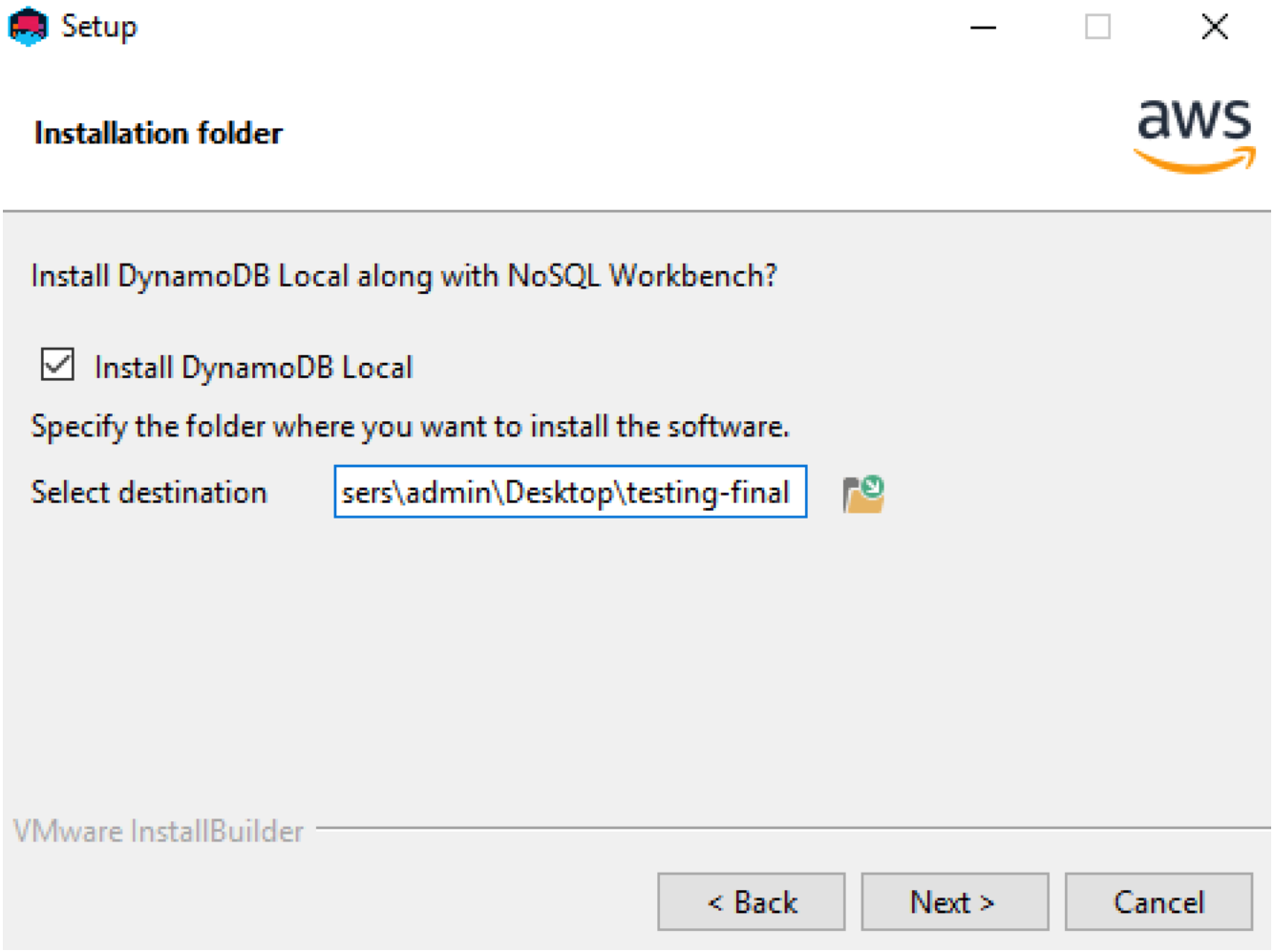
Windows

Para instalar NoSQL Workbench en Windows

1. Ejecute la aplicación del instalador de NoSQL Workbench y elija el idioma de configuración. A continuación, elija OK (Aceptar) para iniciar la configuración. Para obtener más información acerca de cómo descargar NoSQL Workbench, consulte Descargar NoSQL Workbench para DynamoDB.
2. Elija Next (Siguiente) para continuar con la configuración y luego elija Next (Siguiente) en la siguiente pantalla.
3. De forma predeterminada, la casilla de verificación Instalar DynamoDB local está seleccionada para incluir DynamoDB local como parte de la instalación. Al mantener esta opción seleccionada, se garantiza que se instalará DynamoDB local y que la ruta de destino será la misma que la ruta de instalación de NoSQL Workbench. Si se desactiva la casilla de verificación de esta opción, se omitirá la instalación de DynamoDB local y la ruta de instalación será solo para NoSQL Workbench.

   Elija el destino en el que desea instalar el software y elija Next (Siguiente).

   > **Note**
   >
   > Si ha optado por no incluir DynamoDB local como parte de la configuración, desactive la casilla de verificación Instalar DynamoDB local, elija Siguiente y vaya al paso 6. Puede descargar DynamoDB local por separado como instalación independiente más adelante. Para obtener más información, consulte Configuración de la versión de DynamoDB local (versión descargable) .



4. Elija el número de puerto que utilizará DynamoDB local. El puerto predeterminado es 8000. Después de escribir el número de puerto, elija Next (Siguiente).
5. Elija Next (Siguiente) para iniciar la configuración.
6. Cuando se haya completado la configuración, elija Finish (Finalizar) para cerrar la pantalla de configuración.
7. Abra la aplicación en la ruta de instalación, como /programs/DynamoDBWorkbench/.

macOS

Para instalar NoSQL Workbench en macOS

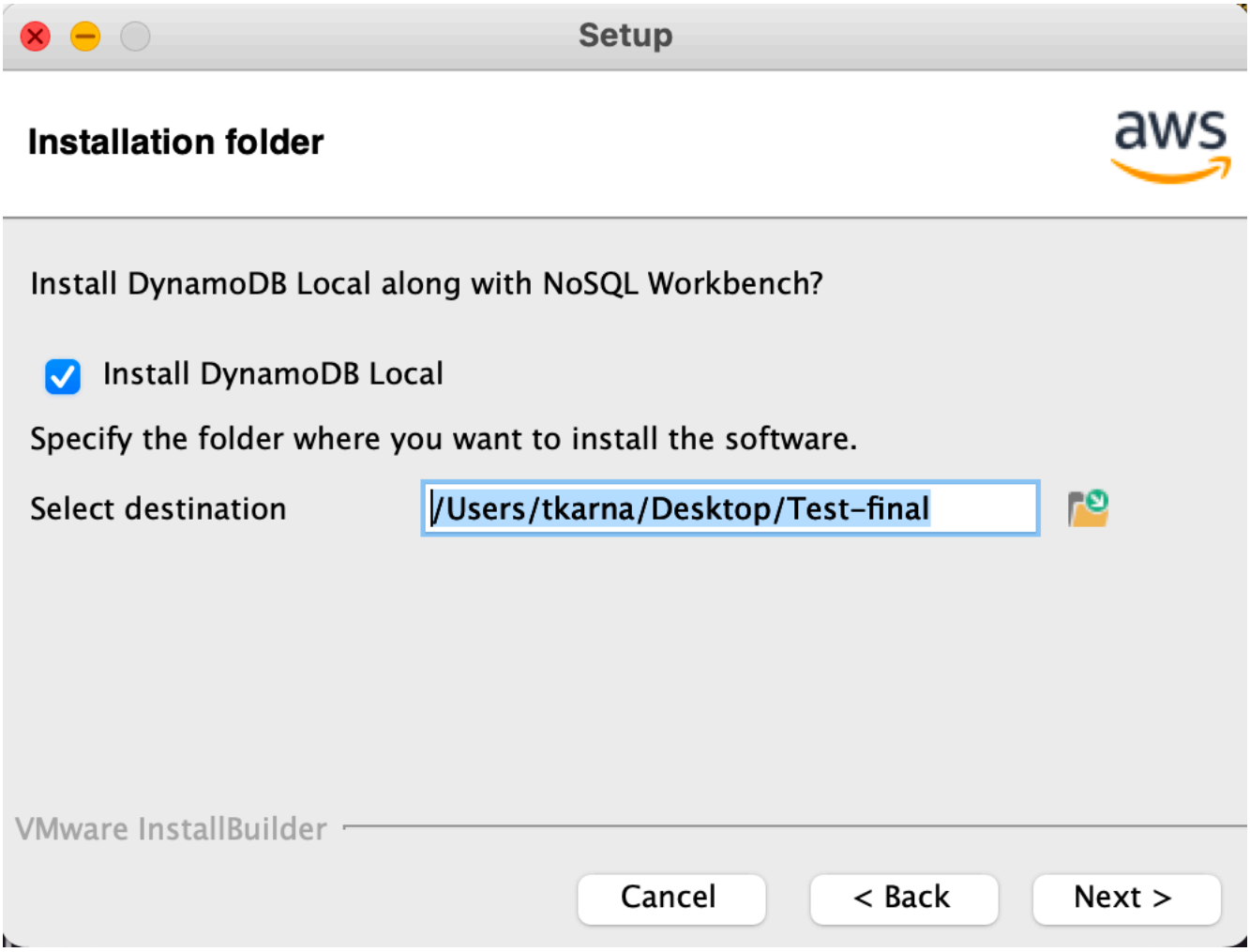
1. Ejecute la aplicación del instalador de NoSQL Workbench y elija el idioma de configuración. A continuación, elija OK (Aceptar) para iniciar la configuración. Para obtener más información

acerca de cómo descargar NoSQL Workbench, consulte Descargar NoSQL Workbench para DynamoDB.

2. Elija Next (Siguiente) para continuar con la configuración y luego elija Next (Siguiente) en la siguiente pantalla.
3. De forma predeterminada, la casilla de verificación Instalar DynamoDB local está seleccionada para incluir DynamoDB local como parte de la instalación. Al mantener esta opción seleccionada, se garantiza que se instalará DynamoDB local y que la ruta de destino será la misma que la ruta de instalación de NoSQL Workbench. Si desactiva esta opción, se omitirá la instalación de DynamoDB local y la ruta de instalación será solo para NoSQL Workbench.

   Elija el destino en el que desea instalar el software y elija Next (Siguiente).

   > **Note**
   >
   > Si ha optado por no incluir DynamoDB local como parte de la configuración, desactive la casilla de verificación Instalar DynamoDB local, elija Siguiente y vaya al paso 6. Puede descargar DynamoDB local por separado como instalación independiente más adelante. Para obtener más información, consulte Configuración de la versión de DynamoDB local (versión descargable) .



4. Elija el número de puerto que utilizará DynamoDB local. El puerto predeterminado es 8000. Después de escribir el número de puerto, elija Next (Siguiente).
5. Elija Next (Siguiente) para iniciar la configuración.
6. Cuando se haya completado la configuración, elija Finish (Finalizar) para cerrar la pantalla de configuración.
7. Abra la aplicación en la ruta de instalación, como /Applications/DynamoDBWorkbench/.

> **Note**
>
> NoSQL Workbench para macOS realiza actualizaciones automáticas. Para recibir notificaciones sobre las actualizaciones, habilite el acceso de las notificaciones a

NoSQL Workbench en System Preferences > Notifications (Preferencias del sistema > Notificaciones).

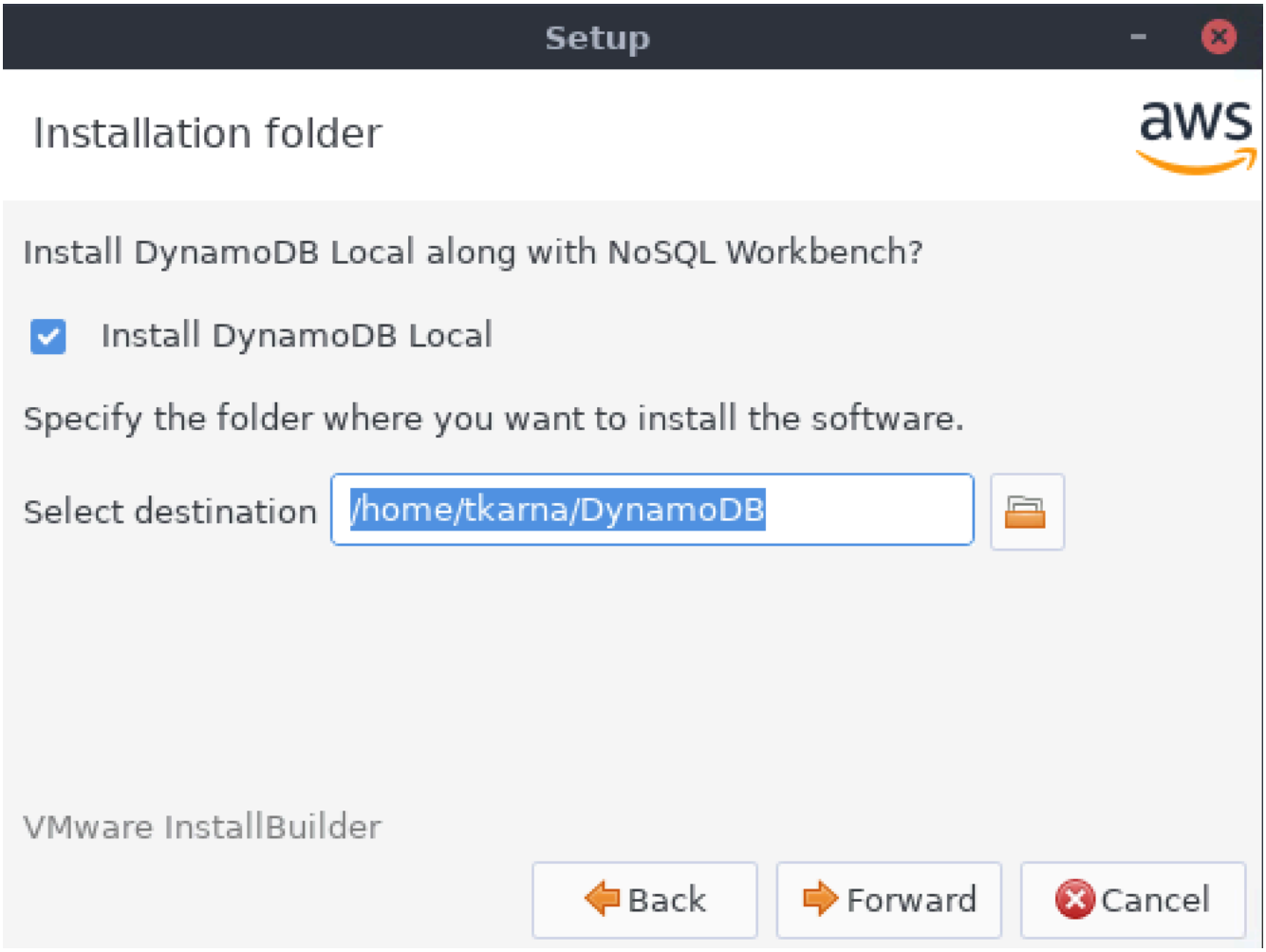
Linux

Para instalar NoSQL Workbench en Linux

1. Ejecute la aplicación del instalador de NoSQL Workbench y elija el idioma de configuración. A continuación, elija OK (Aceptar) para iniciar la configuración. Para obtener más información acerca de cómo descargar NoSQL Workbench, consulte Descargar NoSQL Workbench para DynamoDB.
2. Elija Forward (Reenviar) para continuar con la configuración y elija Forward (Reenviar) en la siguiente pantalla.
3. De forma predeterminada, la casilla de verificación Instalar DynamoDB local está seleccionada para incluir DynamoDB local como parte de la instalación. Al mantener esta opción seleccionada, se garantiza que se instalará DynamoDB local y que la ruta de destino será la misma que la ruta de instalación de NoSQL Workbench. Si desactiva esta opción, se omitirá la instalación de DynamoDB local y la ruta de instalación será solo para NoSQL Workbench.

   Elija el destino en el que desea instalar el software y elija Forward (Reenviar).

   > **Note**
   >
   > Si ha optado por no incluir DynamoDB local como parte de la configuración, desactive la casilla de verificación Instalar DynamoDB local, elija Reenviar y vaya al paso 6. Puede descargar DynamoDB local por separado como instalación independiente más adelante. Para obtener más información, consulte Configuración de la versión de DynamoDB local (versión descargable) .



4. Elija el número de puerto que utilizará DynamoDB local. El puerto predeterminado es 8000. Después de escribir el número de puerto ingresado, elija Forward (Reenviar).
5. Elija Forward (Reenviar) para iniciar la configuración.
6. Cuando se haya completado la configuración, elija Finish (Finalizar) para cerrar la pantalla de configuración.
7. Abra la aplicación en la ruta de instalación, como /usr/local/programs/DynamoDBWorkbench/.

Inicio de una AppImage en Linux

1. Convertir el archivo de AppImage en ejecutable:

   chmod +x noSQL-workbench-linux.AppImage

Reemplace noSQL-workbench-linux.AppImage por el nombre real del archivo de la AppImage que ha descargado.

2. Ejecutar AppImage:

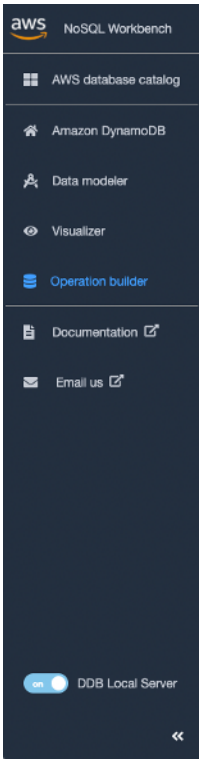
   ./noSQL-workbench-linux.AppImage

   Esto iniciará la aplicación NoSQL Workbench.

> **Note**
>
> Según la distribución de Linux que utilice, es posible que necesite instalar dependencias adicionales para que AppImage funcione correctamente. Si tiene algún problema, consulte la documentación proporcionada por los desarrolladores de AppImage o busque ayuda en la comunidad.

> **Note**
>
> Si optó por instalar DynamoDB local como parte de la instalación de NoSQL Workbench, DynamoDB local se preconfigurará con las opciones predeterminadas. Para editar las opciones predeterminadas, modifique el script DDBLocalStart que se encuentra en el directorio /resources/DDBLocal_Scripts/. Puede encontrarlo en la ruta que proporcionó durante la instalación. Para obtener más información sobre las opciones de DynamoDB local, consulte Notas sobre el uso local de DynamoDB .

Si optó por instalar DynamoDB local como parte de la instalación de NoSQL Workbench, tendrá acceso a un botón para habilitar y desactivar DynamoDB local, como se muestra en la siguiente imagen.



Creación de modelos de datos con NoSQL Workbench

Puede utilizar la herramienta del modelador de datos en NoSQL Workbench para Amazon DynamoDB con el fin de crear nuevos modelos de datos o para diseñar modelos basados en otros existentes que satisfagan los patrones de acceso a datos de sus aplicaciones. El modelador de datos incluye algunos modelos de datos de muestra para ayudarle a comenzar.

Temas

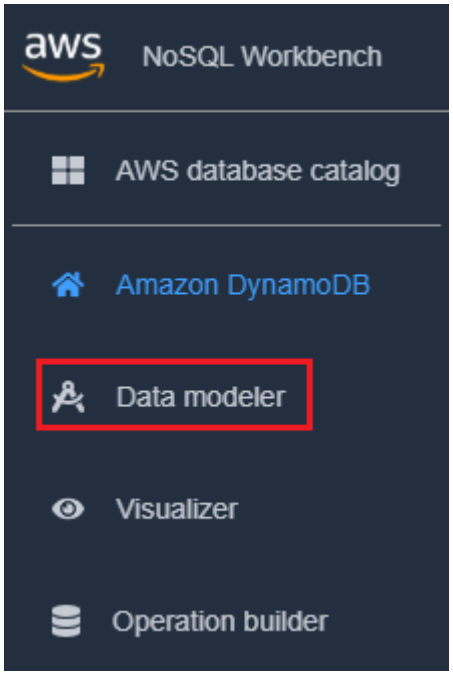
- Creación de un nuevo modelo de datos
- Importación de un modelo de datos existente
- Exportación de un modelo de datos
- Edición de un modelo de datos existente

Creación de un nuevo modelo de datos

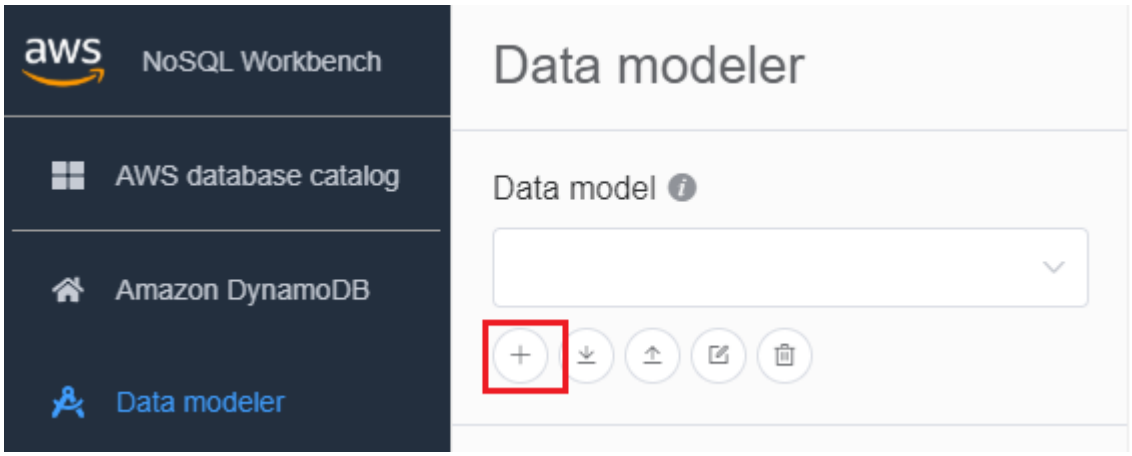
Siga estos pasos para crear un nuevo modelo de datos en Amazon DynamoDB utilizando NoSQL Workbench.

Para crear un nuevo modelo de datos

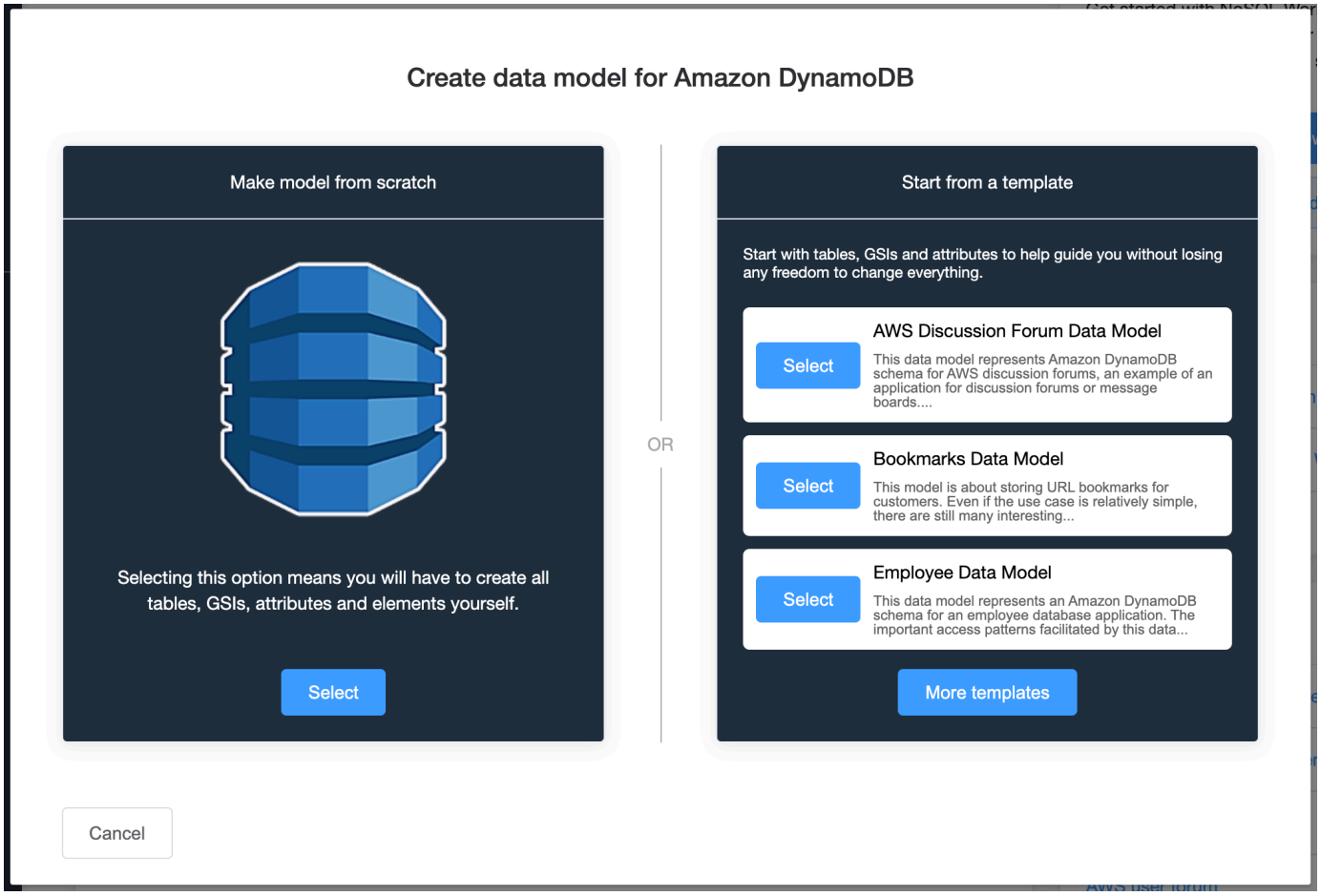
1. Abra NoSQL Workbench y, en el panel de navegación de la izquierda, elija el icono Data modeler (Modelador de datos).



2. Elija Create data model (Crear modelo de datos).

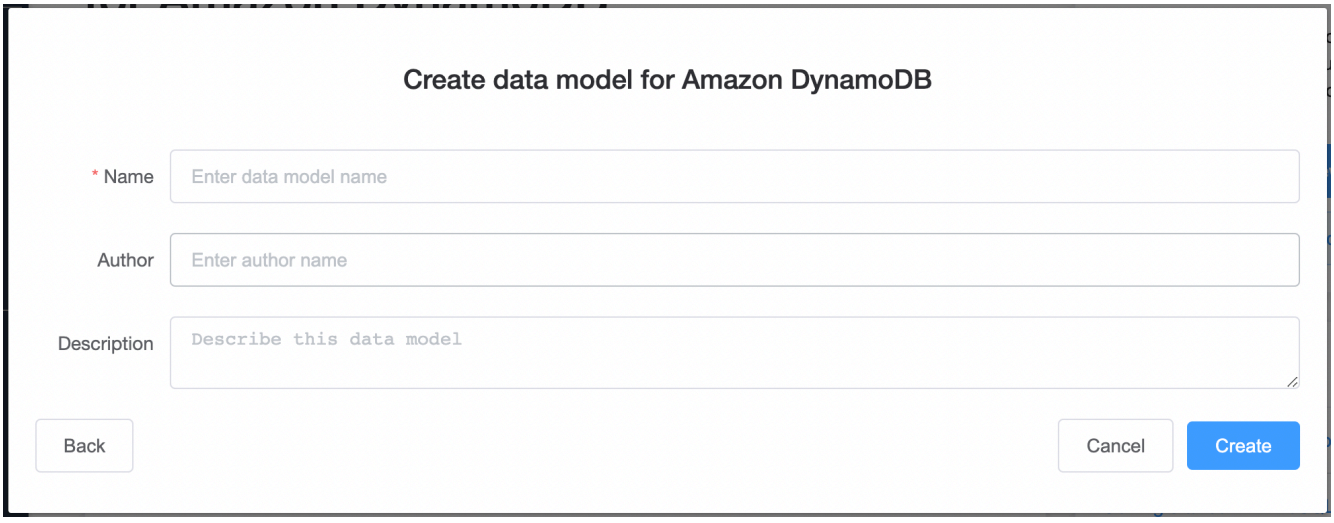


Create data model (Crear modelo de datos) tiene dos opciones: crear un modelo de cero y comenzar desde una plantilla.



Make model from scratch

Para crear un modelo de cero, escriba un nombre, un autor y una descripción para el modelo de datos. Elija Create (Crear) cuando termine.



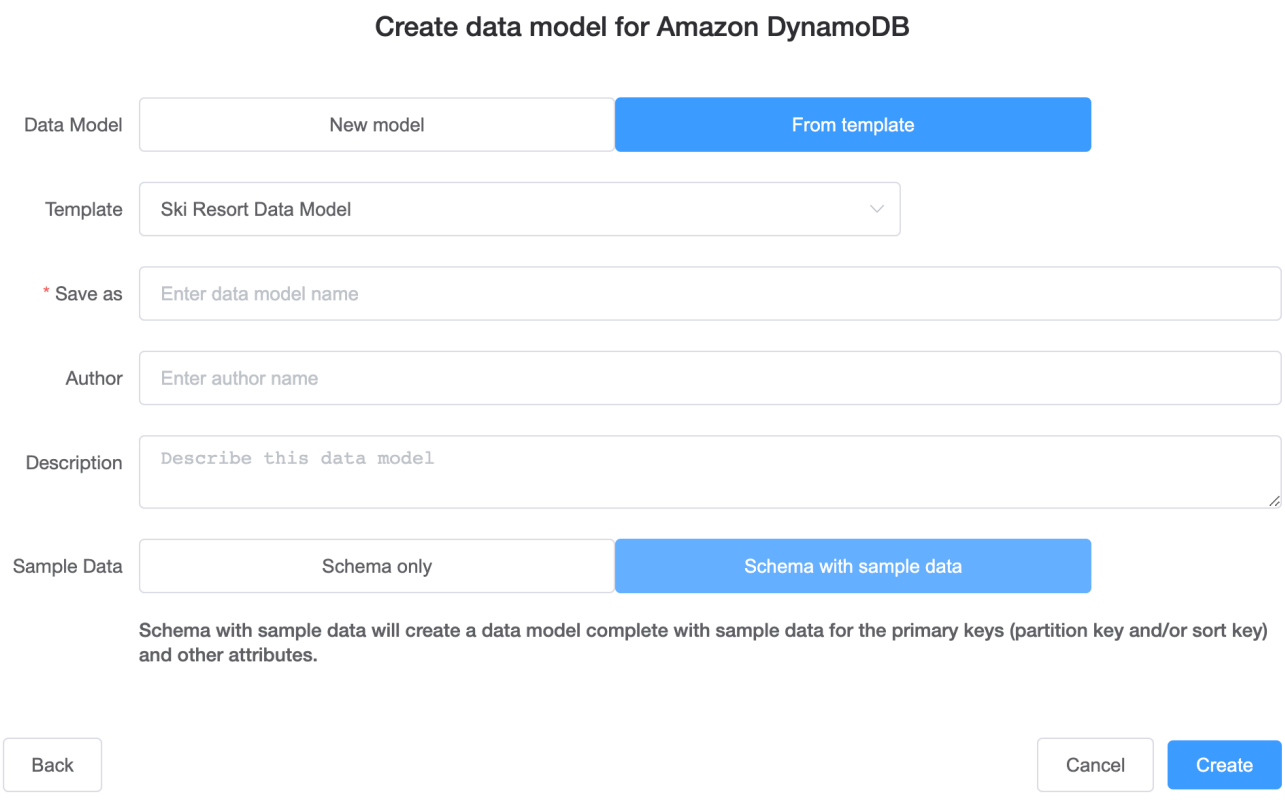
Start from a template

Partir de una plantilla le permite elegir un modelo de ejemplo desde el que empezar. Elija More templates (Más plantillas) para ver más opciones de plantillas. Elija Select (Seleccionar) para la plantilla que desea usar.

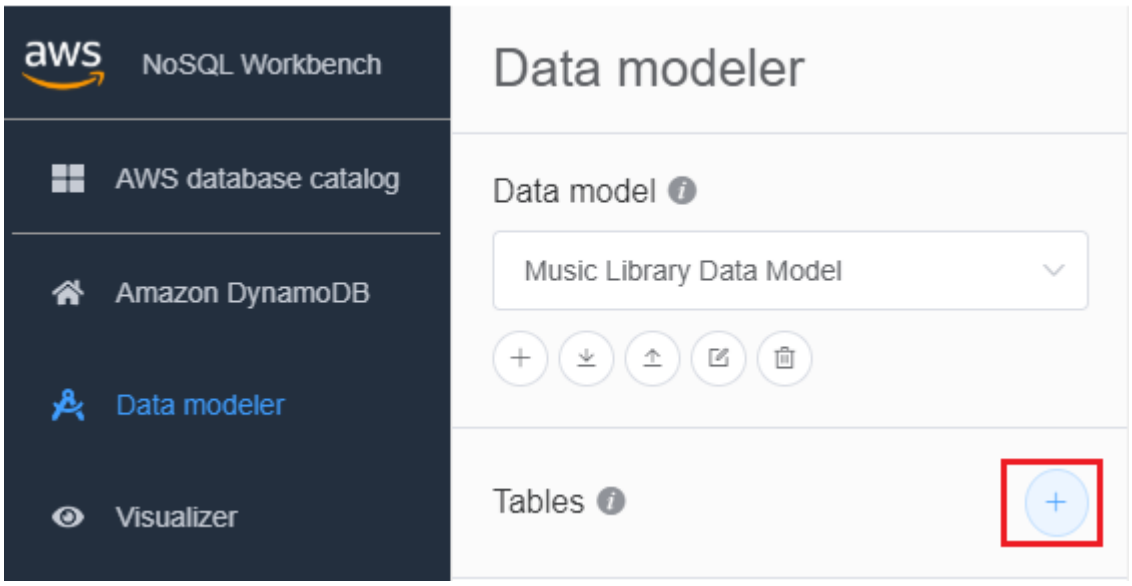
Escriba el nombre, el autor y la descripción de un modelo de datos para la plantilla que ha seleccionado. Puede elegir entre Schema only (Solo esquema) y Schema with sample data (Esquema con datos de ejemplo).

- Schema only (Solo esquema) crea un modelo de datos vacío con la clave principal (clave de partición y clasificación) y otros atributos.
- Schema with sample data (Esquema con datos de ejemplo) creará un modelo de datos completo con datos de ejemplo para la clave principal (clave de partición y clasificación) y otros atributos.

Cuando se complete esta información, elija Create (Crear) para crear el modelo.



3. Con el modelo creado, elija Add table (Agregar tabla).



Para obtener más información sobre las tablas, consulte Uso de tablas en DynamoDB.

4. Especifique lo siguiente:

   - Table name (Nombre de tabla): escriba un nombre único para la tabla.
   - Clave de partición: ingrese un nombre de clave de partición y especifique su tipo. Opcionalmente, también puede seleccionar un formato de tipo de datos más granular para la generación de datos de muestra.
   - Si desea añadir una clave de ordenación:
     1. Seleccione Add sort key (Añadir clave de ordenación).
     2. Especifique el nombre de la clave de ordenación y su tipo. Opcionalmente, puede seleccionar un formato de tipo de datos más granular para la generación de datos de muestra.

> **Note**
>
> Para obtener más información sobre el diseño de la clave principal, el diseño y el uso eficaz de las claves de partición y el uso de las claves de clasificación, consulte lo siguiente:
>
> - Clave principal
> - Prácticas recomendadas para diseñar claves de partición y utilizarlas con eficacia en DynamoDB

- Prácticas recomendadas sobre el uso de claves de clasificación para organizar datos en DynamoDB

5. Para añadir otros atributos, haga lo siguiente por cada uno:

   1. Elija Agregar atributo.
   2. Especifique el nombre de atributo y su tipo. Opcionalmente, puede seleccionar un formato de tipo de datos más granular para la generación de datos de muestra.

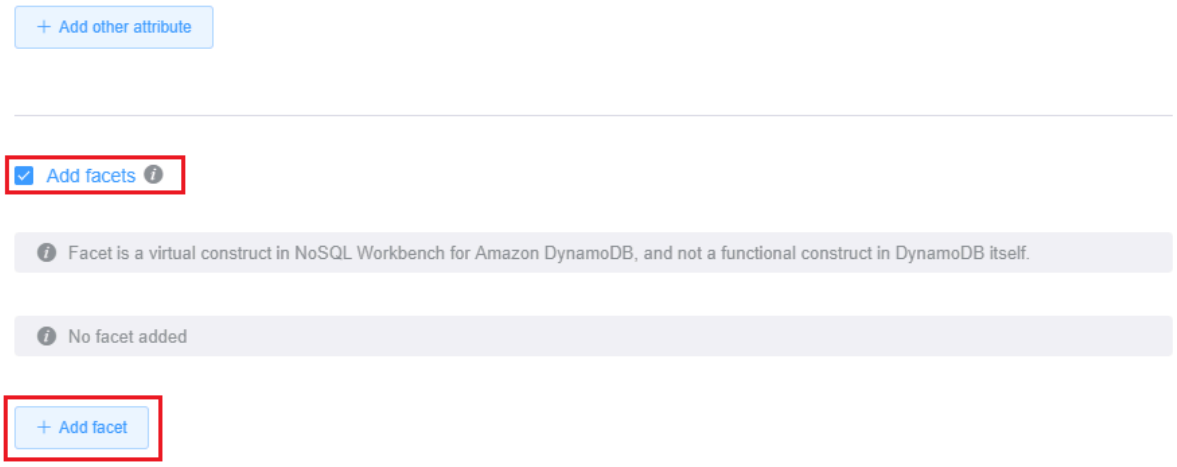
6. Agregar una faceta:

   Si lo desea, puede agregar facetas. Una faceta es un constructo virtual en NoSQL Workbench. No es un constructo funcional en el propio DynamoDB.

   > **Note**
   >
   > Las facetas en NoSQL Workbench le ayudan a visualizar los diferentes patrones de acceso a los datos de una aplicación para Amazon DynamoDB con solo un subconjunto de los datos de una tabla. Para obtener más información sobre facetas, consulte Visualización de patrones de acceso a datos.

   Para agregar una faceta,

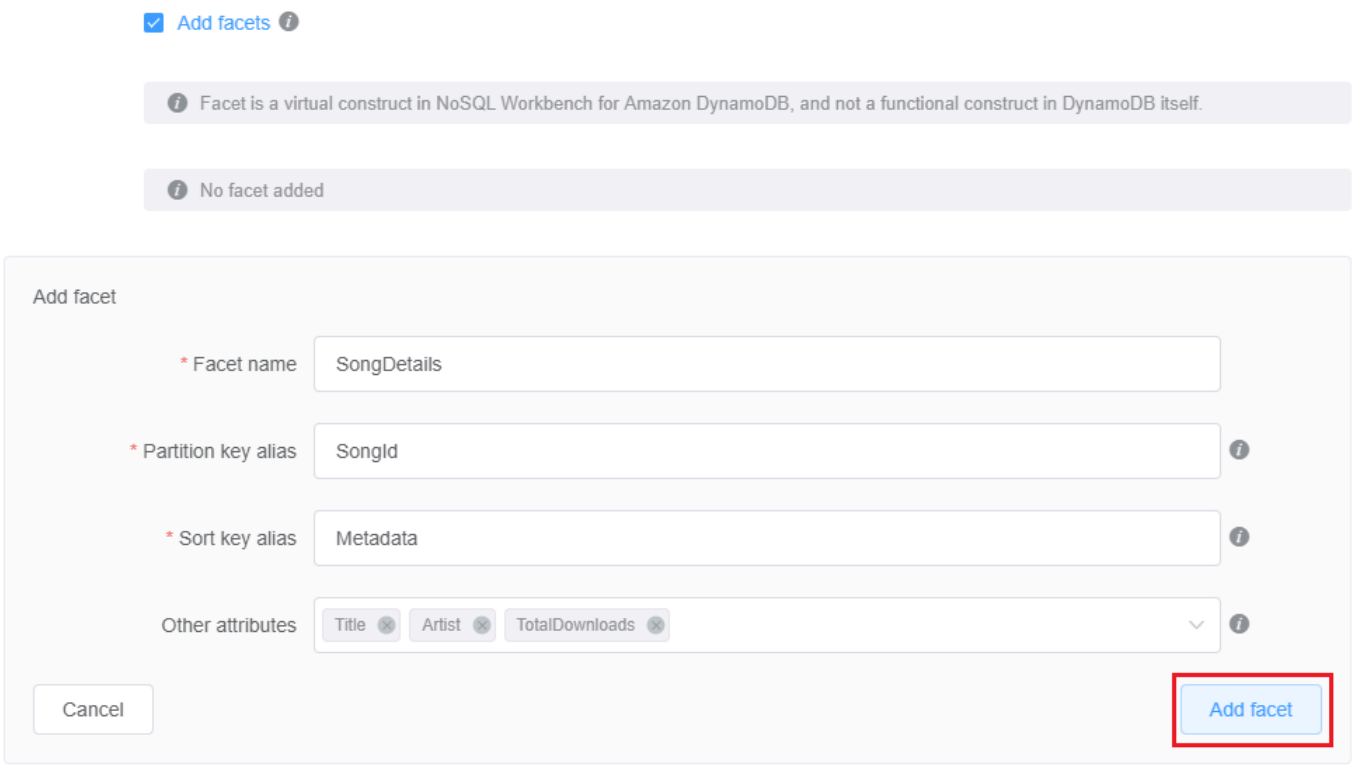
   - Seleccione Add facets (Agregar facetas).
   - Elija Add facet (Agregar faceta).



   - Especifique lo siguiente:

- El nombre de la faceta en Facet name.
- Un alias de clave de partición para distinguir esta vista de faceta.
- Un alias de clave de ordenación en Sort key alias.
- Seleccione los Other attributes (Otros atributos) que forman parte de esta faceta.

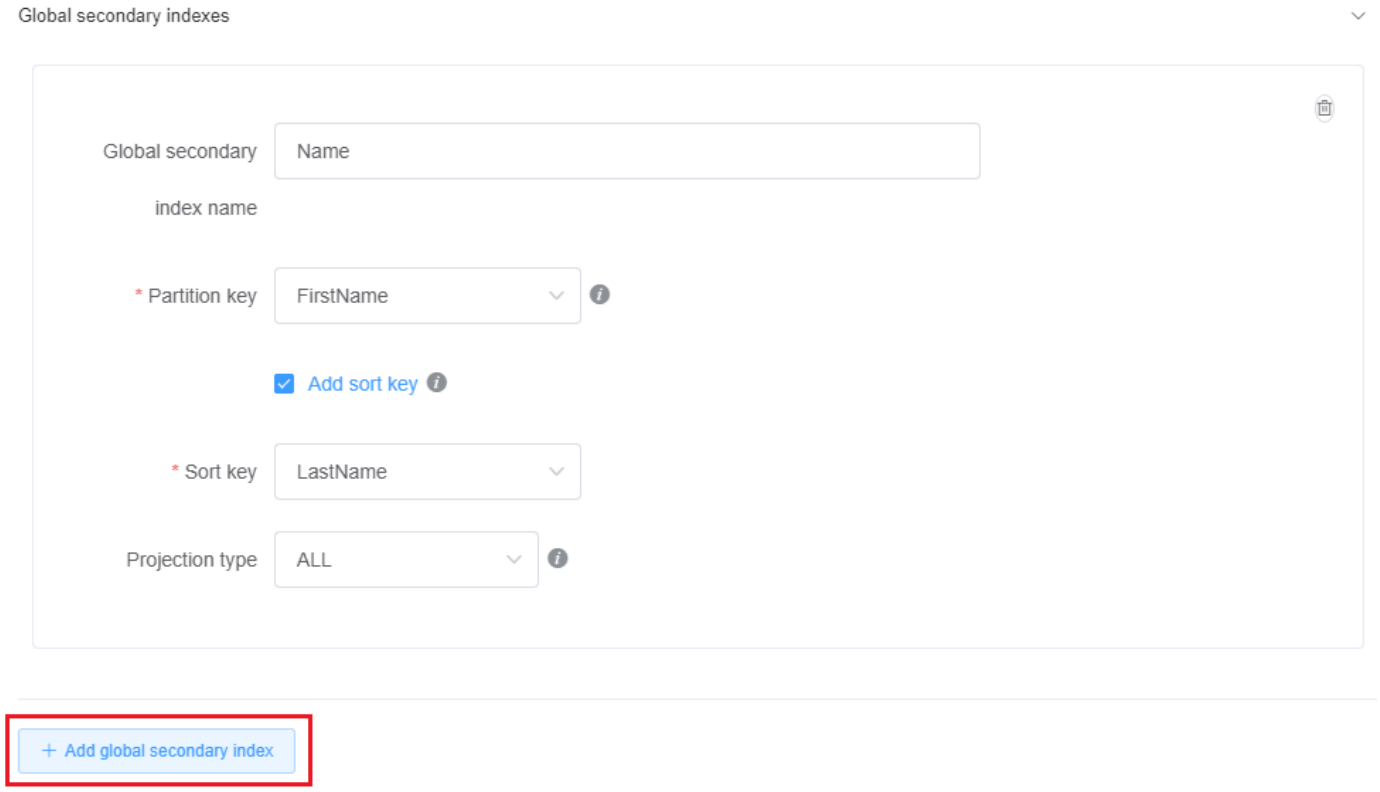
Elija Add facet (Agregar faceta).



Repita este paso si desea agregar más facetas.

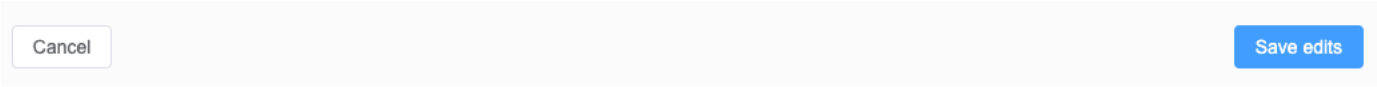
7. Si desea añadir un índice secundario global, elija Add global secondary index (Añadir índice secundario global).

   Especifique los valores de Global secondary index name (Nombre de índice secundario global), el atributo Partition key (Clave de partición) y Projection type (Tipo de partición).



Para obtener más información sobre cómo trabajar con índices secundarios globales en DynamoDB, consulte Índices secundarios globales.

8. Guarde las modificaciones en la configuración de la tabla.



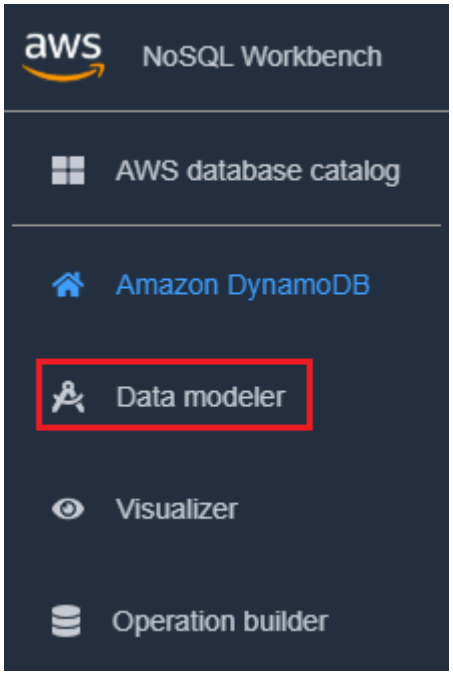
Para obtener más información, consulte sobre la operación de la API `CreateTable`, consulte Crear tabla en la Referencia de la API de Amazon DynamoDB.

Importación de un modelo de datos existente

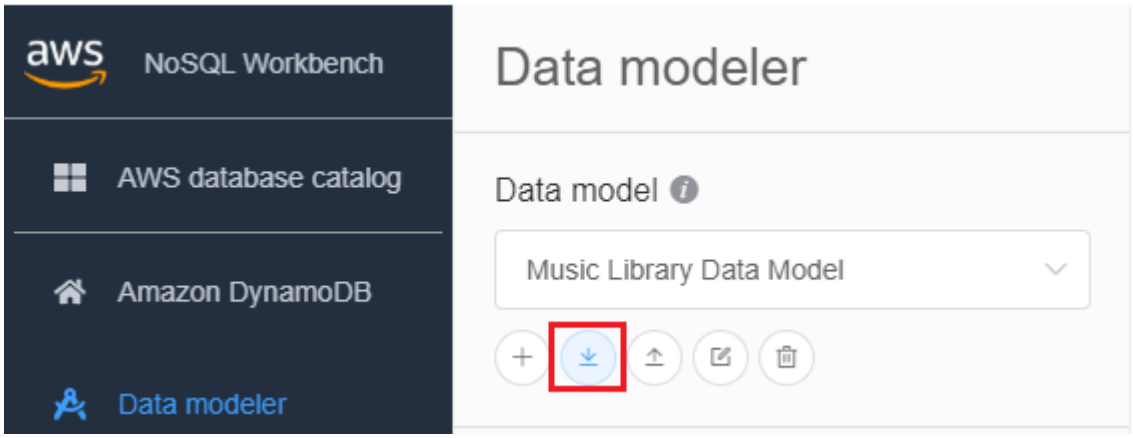
Puede utilizar NoSQL Workbench para Amazon DynamoDB con el fin de crear un modelo de datos mediante la importación y la modificación de un modelo existente. Puede importar modelos de datos en formato de modelo de NoSQL Workbench o en Formato de plantilla JSON de AWS CloudFormation.

Para importar un modelo de datos

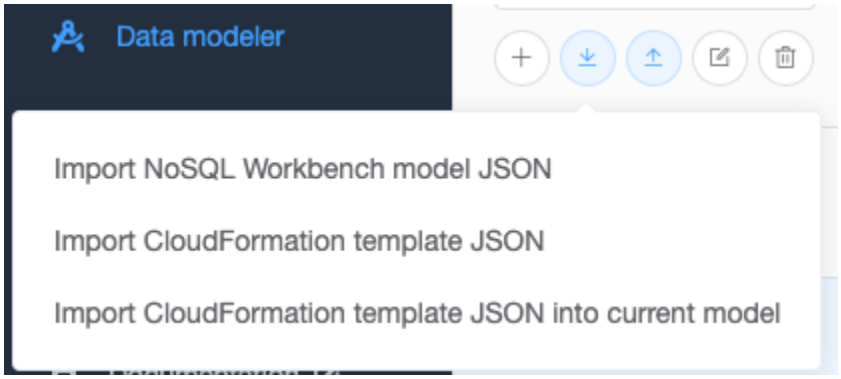
1. En NoSQL Workbench, en el panel de navegación de la izquierda, seleccione el icono Data modeler (Modelador de datos).



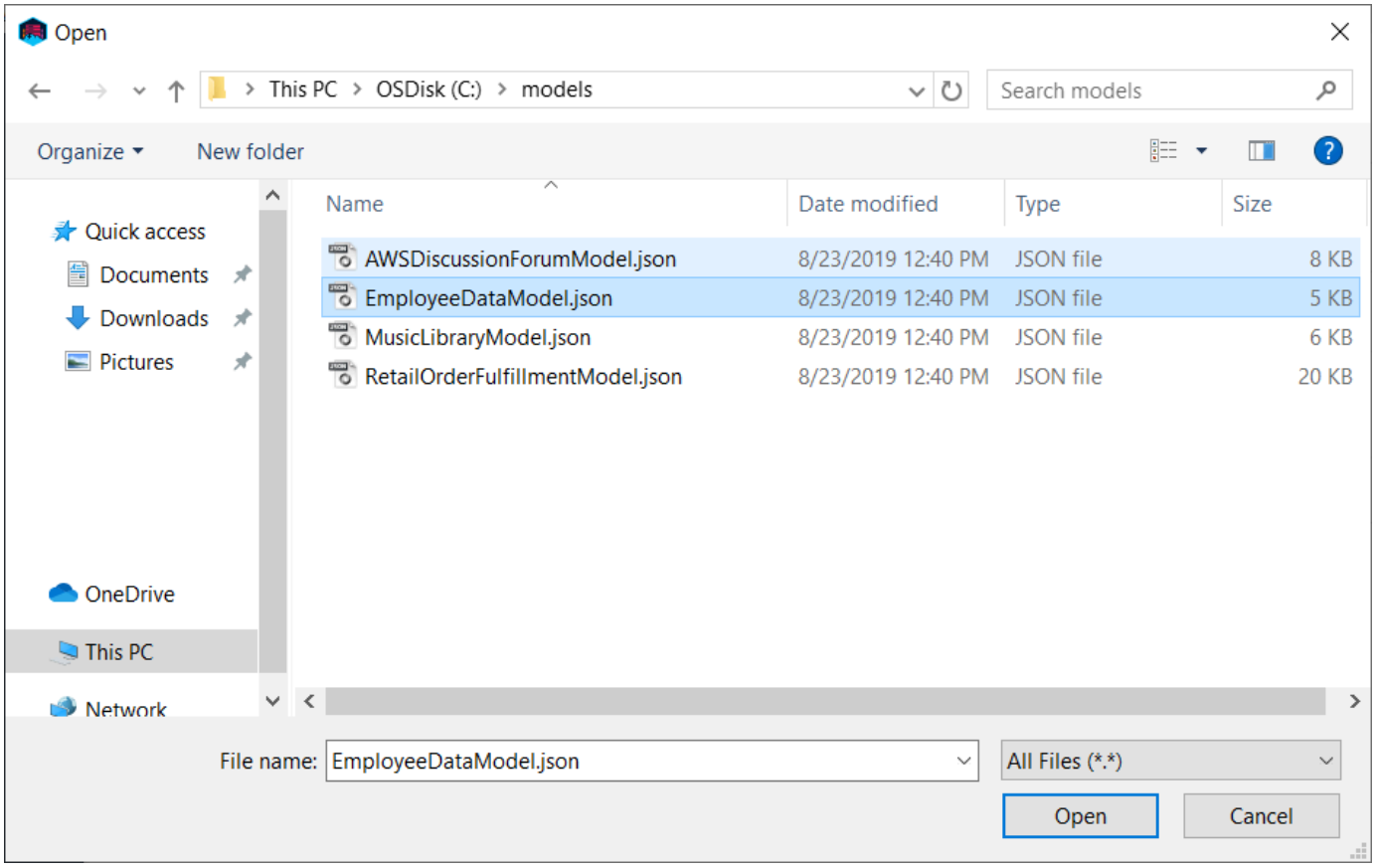
2. Pase el puntero sobre Import data model (Importar modelo de datos).



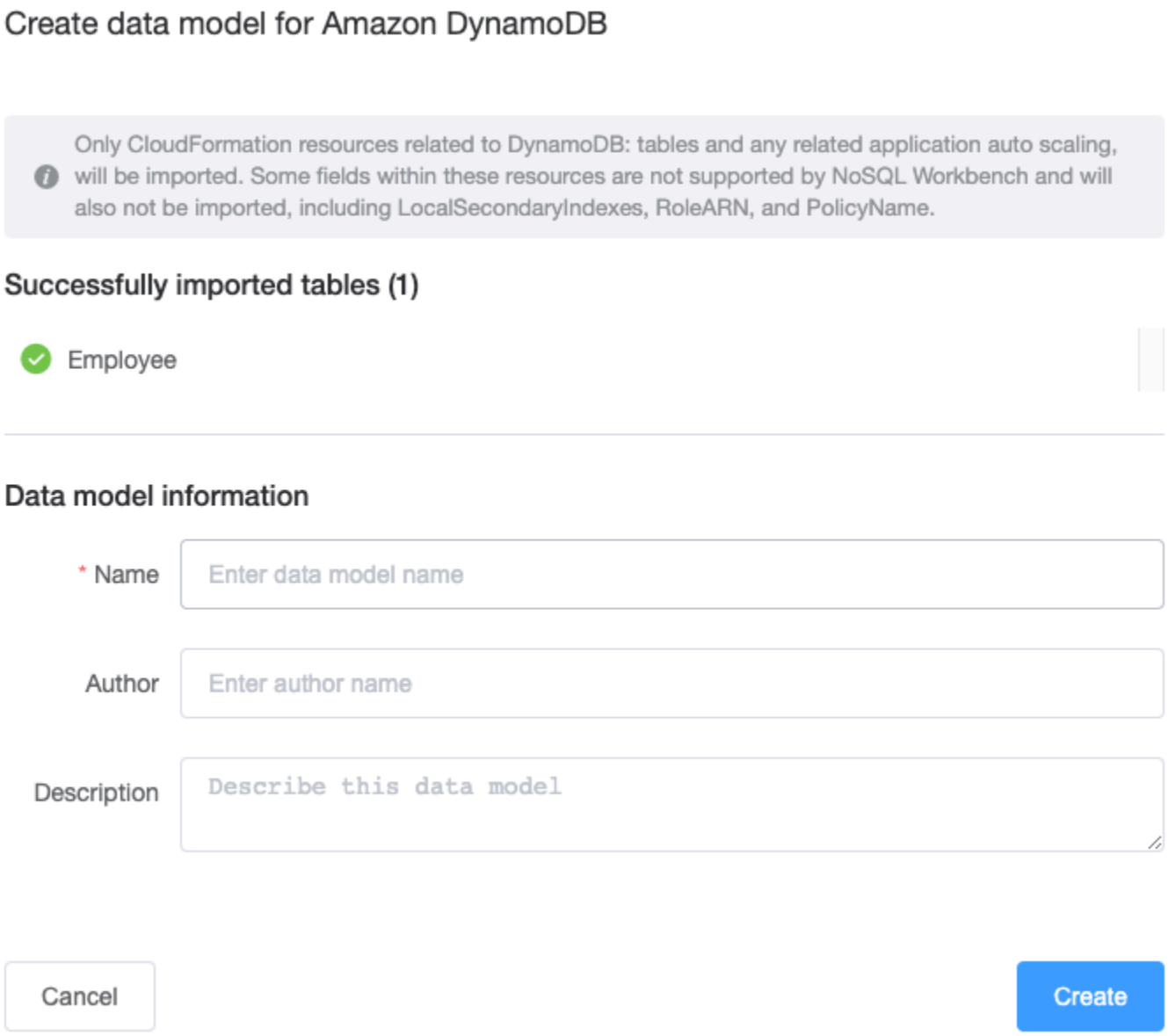
   En la lista desplegable, elija si el modelo que desea importar está en formato de modelo NoSQL Workbench o en formato de plantilla JSON de CloudFormation. Si tiene abierto un modelo de datos existente en NoSQL Workbench, tendrá la opción de importar una plantilla de CloudFormation al modelo actual.



3. Elija un modelo para importar.



4. Si el modelo que va a importar está en formato de plantilla de CloudFormation, verá una lista de tablas que se van a importar y tendrá la oportunidad de especificar un nombre, un autor y una descripción del modelo de datos.

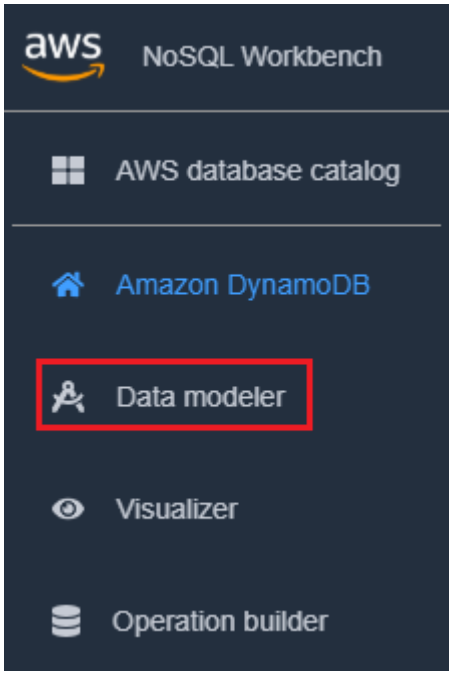


Exportación de un modelo de datos

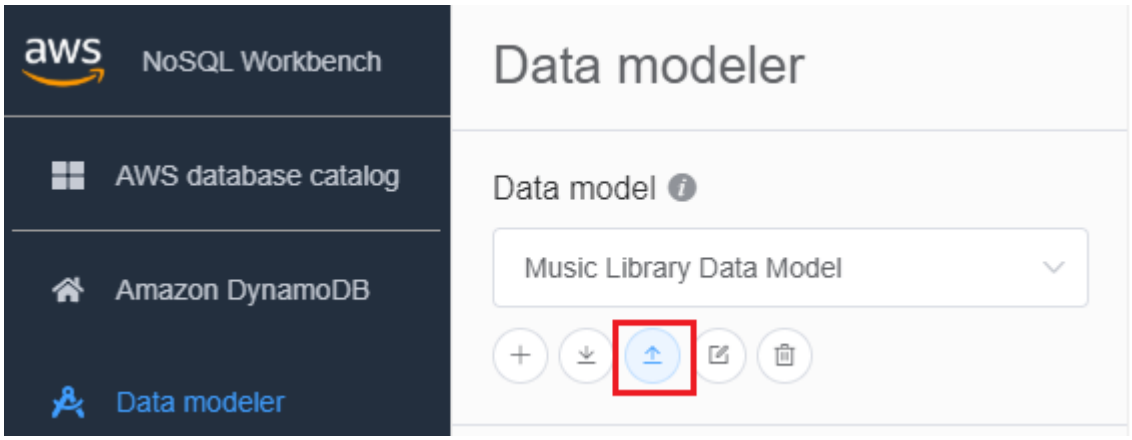
Después de crear un modelo de datos utilizando NoSQL Workbench para Amazon DynamoDB, puede guardarlo y exportarlo en formato de modelo NoSQL Workbench o formato de plantilla JSON AWS CloudFormation.

Para exportar un modelo de datos

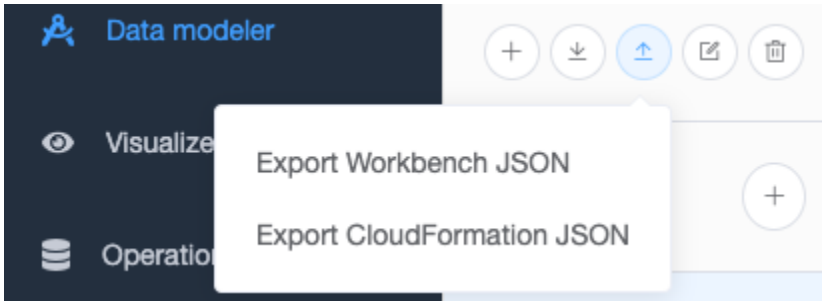
1. En NoSQL Workbench, en el panel de navegación de la izquierda, seleccione el icono Data modeler (Modelador de datos).



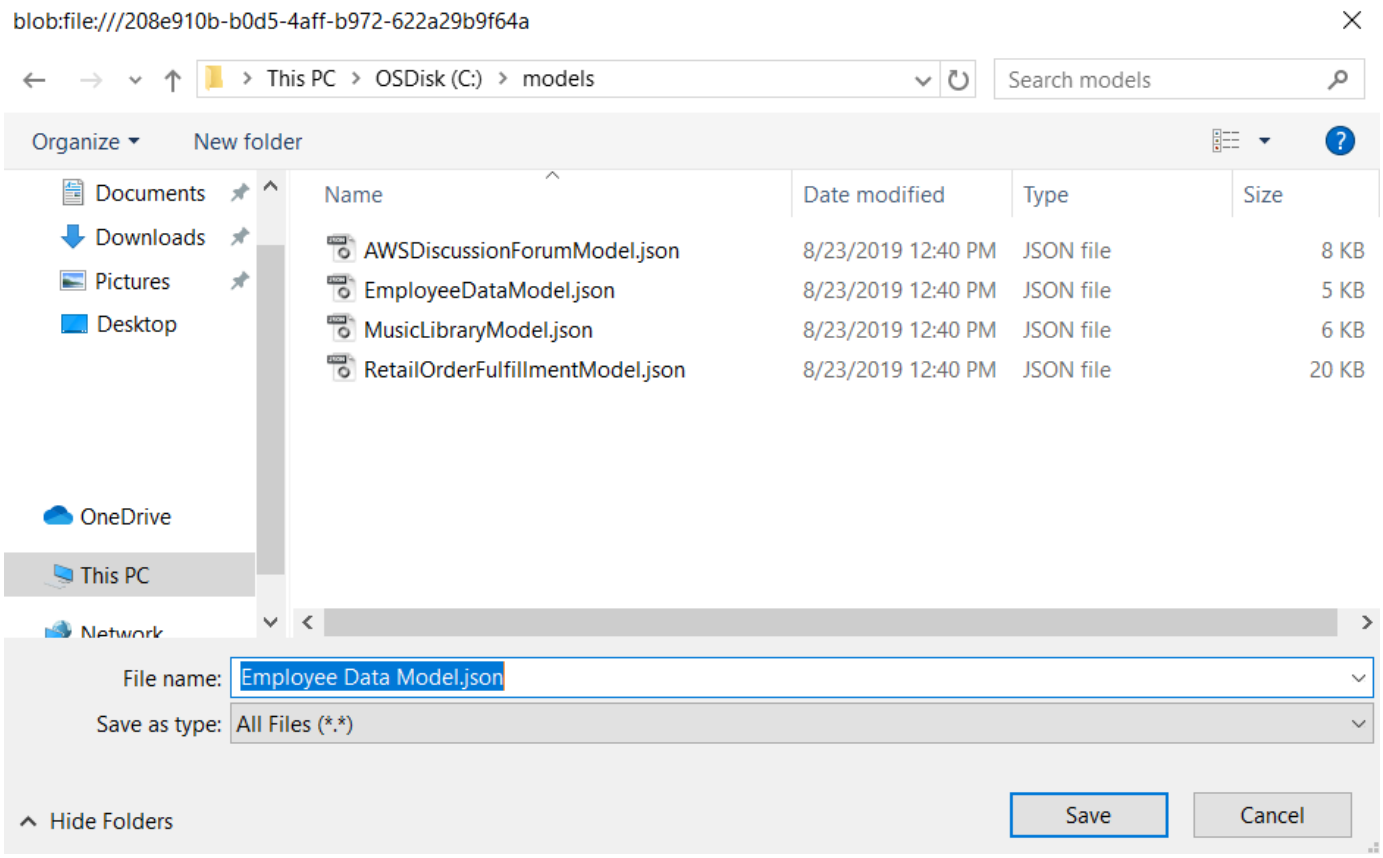
2. Pase el puntero sobre Export data model (Exportar modelo de datos).



En la lista desplegable, elija si desea exportar el modelo de datos en formato de modelo de NoSQL Workbench o en formato de plantilla JSON de CloudFormation.



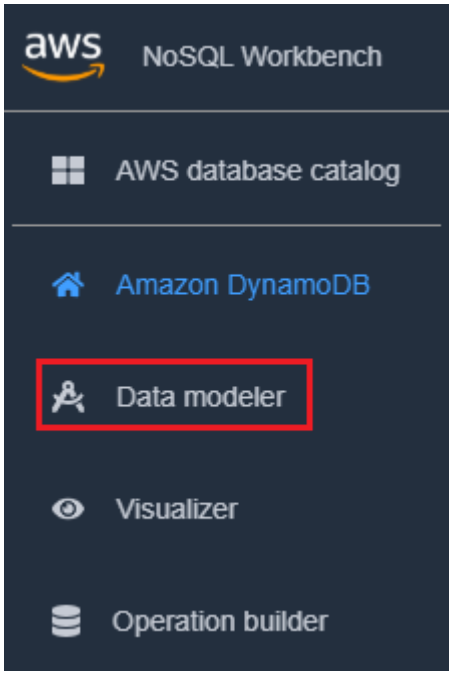
3. Elija una ubicación para guardar el modelo.



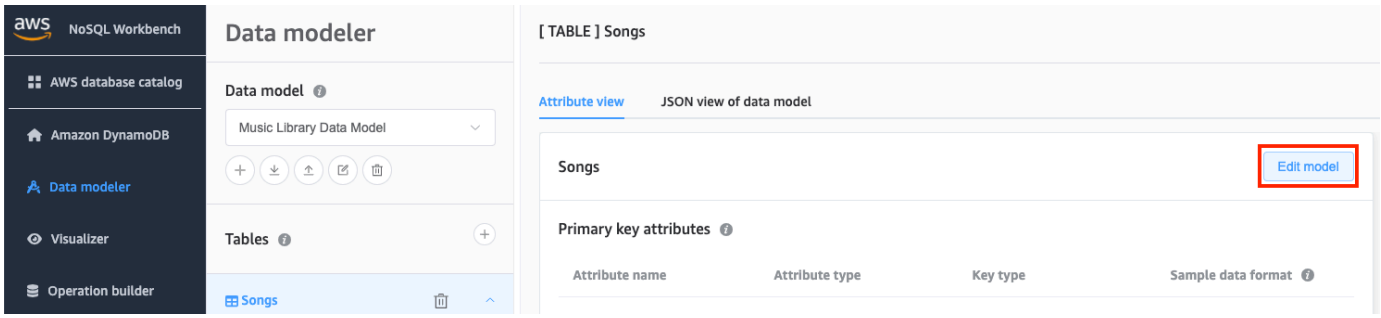
Edición de un modelo de datos existente

Para editar un modelo existente

1. En NoSQL Workbench, en el panel de navegación de la izquierda, seleccione el botón Data modeler (Modelador de datos).



2. Seleccione el modelo de datos y elija la tabla que desee editar. Elija Editar modelo.



3. Realice las ediciones necesarias y, a continuación, seleccione Save edits (Guardar ediciones).

Para editar manualmente un modelo existente y agregar una faceta

1. Exporte el modelo. Para obtener más información, consulte Exportación de un modelo de datos.
2. Abra el archivo exportado en un editor.
3. Localice el objeto `DataModel` de la tabla para la que desea crear una faceta.

   Añada una matriz `TableFacets` que represente todas las facetas de la tabla.

   Para cada faceta, agregue un objeto a la matriz `TableFacets`. Cada elemento de la matriz tiene las siguientes propiedades:

   - `FacetName`: un nombre para la faceta. Este valor debe ser único en todo el modelo.

- `PartitionKeyAlias`: un nombre fácil de mencionar para la clave de partición de la tabla. Este alias se muestra cuando visualiza la faceta en NoSQL Workbench.
- `SortKeyAlias`: un nombre fácil de mencionar para la clave de clasificación de la tabla. Este alias se muestra cuando visualiza la faceta en NoSQL Workbench. Esta propiedad no es necesaria si la tabla no tiene definida ninguna clave de clasificación.
- `NonKeyAttributes`: una matriz de nombres de atributos que se necesitan para el patrón de acceso. Estos nombres deben mapearse a los nombres de atributo definidos para la tabla.

```
{
  "ModelName": "Music Library Data Model",
  "DataModel": [
    {
      "TableName": "Songs",
      "KeyAttributes": {
        "PartitionKey": {
          "AttributeName": "Id",
          "AttributeType": "S"
        },
        "SortKey": {
          "AttributeName": "Metadata",
          "AttributeType": "S"
        }
      },
      "NonKeyAttributes": [
        {
          "AttributeName": "DownloadMonth",
          "AttributeType": "S"
        },
        {
          "AttributeName": "TotalDownloadsInMonth",
          "AttributeType": "S"
        },
        {
          "AttributeName": "Title",
          "AttributeType": "S"
        },
        {
          "AttributeName": "Artist",
          "AttributeType": "S"
        },
        {
```

```
          "AttributeName": "TotalDownloads",
          "AttributeType": "S"
        },
        {
          "AttributeName": "DownloadTimestamp",
          "AttributeType": "S"
        }
      ],
      "TableFacets": [
        {
          "FacetName": "SongDetails",
          "KeyAttributeAlias": {
            "PartitionKeyAlias": "SongId",
            "SortKeyAlias": "Metadata"
          },
          "NonKeyAttributes": [
            "Title",
            "Artist",
            "TotalDownloads"
          ]
        },
        {
          "FacetName": "Downloads",
          "KeyAttributeAlias": {
            "PartitionKeyAlias": "SongId",
            "SortKeyAlias": "Metadata"
          },
          "NonKeyAttributes": [
            "DownloadTimestamp"
          ]
        }
      ]
    }
  ]
}
```

4. Ahora puede importar el modelo modificado a NoSQL Workbench. Para obtener más información, consulte Importación de un modelo de datos existente.

Visualización de patrones de acceso a datos

Puede utilizar la herramienta de visualización en NoSQL Workbench para Amazon DynamoDB con el fin de mapear consultas y visualizar diferentes patrones de acceso (conocidos como facetas) de una aplicación. Cada faceta corresponde a un patrón de acceso diferente en DynamoDB. También puede añadir datos manualmente a su modelo de datos o importar datos desde MySQL.

Temas

- Incorporación de datos de muestreo a un modelo de datos
- Importación de datos de muestra de un archivo CSV
- Visualización de patrones de acceso a datos
- Visualización de todas las tablas de un modelo de datos mediante la vista agregada
- Confirmación de un modelo de datos en DynamoDB

Incorporación de datos de muestreo a un modelo de datos

Al agregar datos de muestra a su modelo, puede mostrar datos al visualizar el modelo y sus diversos patrones de acceso a datos o facetas.

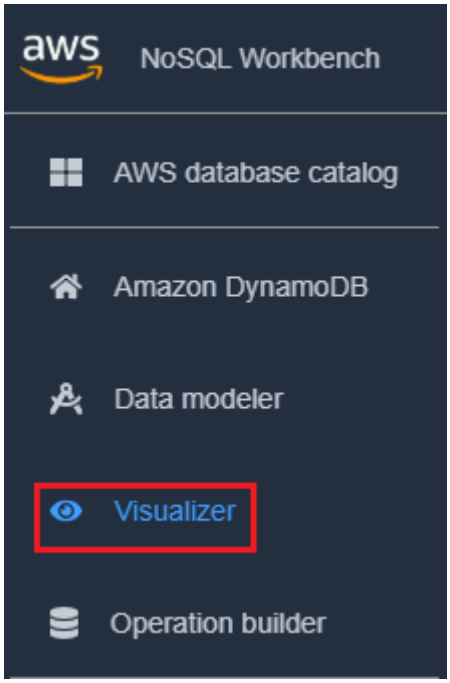
Hay dos formas de agregar datos de muestra. Una de ellas es utilizar nuestra herramienta de generación automática de datos de muestra. La otra es agregar los datos de uno en uno.

Siga estos pasos para añadir datos de muestra a un modelo de datos utilizando NoSQL Workbench para Amazon DynamoDB.

Para generar automáticamente datos de muestra

La generación automática de datos de muestra le ayuda a generar entre 1 y 5000 filas de datos para su utilización inmediata. Puede especificar un tipo de datos de muestra granular para crear datos realistas basados en sus necesidades de diseño y pruebas. Para utilizar la capacidad de generar datos realistas, debe especificar el formato del tipo de datos de muestra para sus atributos en el modelador de datos. Consulte Creación de un nuevo modelo de datos para especificar ejemplos de formatos de tipos de datos.

1. En el panel de navegación de la izquierda, elija el icono visualizer (visualizador).

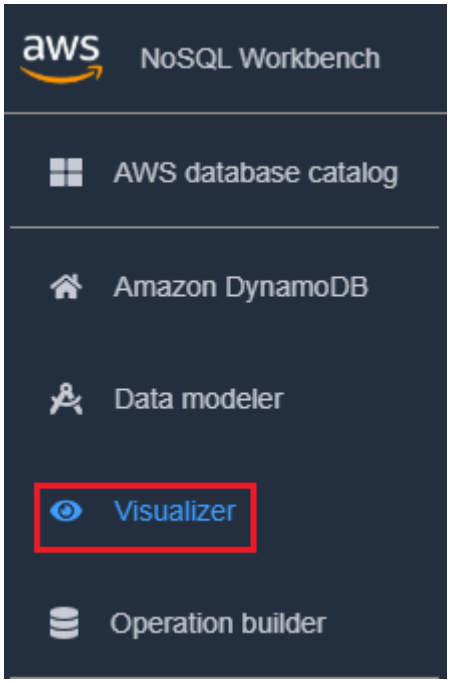


2. En el visualizador, seleccione el modelo de datos y elija la tabla.
3. Elija el menú desplegable Acción y seleccione Agregar datos de muestra.

4. Introduzca el número o los elementos de datos de muestra que desea generar y, a continuación, seleccione Confirmar.

Para agregar datos de muestra de uno en uno

1. En el panel de navegación de la izquierda, elija el icono visualizer (visualizador).

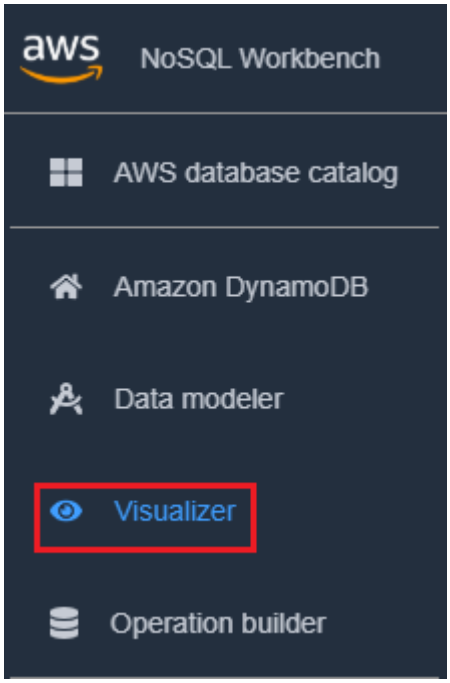


2. En el visualizador, seleccione el modelo de datos y elija la tabla.
3. Elija el menú desplegable Acción y seleccione Editar datos.

4. Elija Agregar nueva regla. Escriba los datos de ejemplo en los campos de texto vacíos y elija Agregar nueva fila de nuevo para agregar filas adicionales. Cuando termine, elija Guardar cambios.

Para eliminar datos de muestra

1. En el panel de navegación de la izquierda, elija el icono visualizer (visualizador).



2. En el visualizador, seleccione el modelo de datos y elija la tabla.
3. Elija el menú desplegable Acción y seleccione Editar datos.

4. Seleccione el icono de eliminación situado junto a cada fila de datos que desee eliminar.

Importación de datos de muestra de un archivo CSV

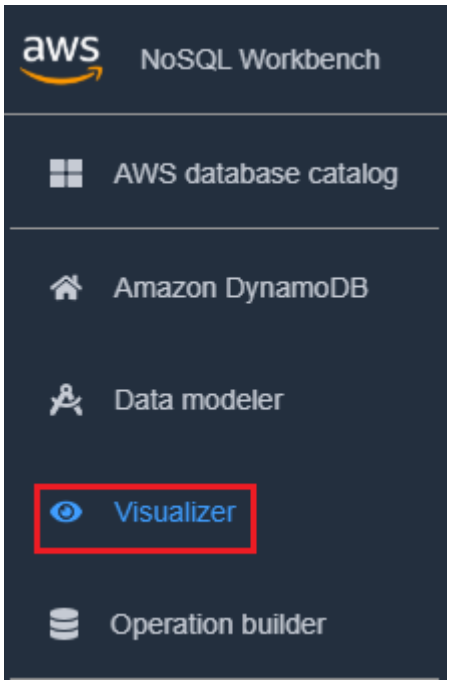
Si dispone de datos de muestra preexistentes en un archivo CSV, puede importarlos a NoSQL Workbench. Esto le permite rellenar rápidamente el modelo con datos de muestra sin tener que ingresarlos línea por línea.

Los nombres de columna del archivo CSV deben coincidir con los nombres de atributos del modelo de datos, pero no es necesario que estén en el mismo orden. Por ejemplo, si el modelo de datos tiene atributos denominados `LoginAlias`, `FirstName`, y `LastName`, las columnas CSV podrían ser `LastName`, `FirstName`, y `LoginAlias`.

La importación de datos desde un archivo CSV se limita a 150 filas a la vez.

Para importar datos de un archivo CSV a NoSQL Workbench

1. En el panel de navegación de la izquierda, elija el icono visualizer (visualizador).



2. En el visualizador, seleccione el modelo de datos y elija la tabla.
3. Elija el menú desplegable Acción y seleccione Editar datos.
4. Elija el menú desplegable Acción de nuevo y seleccione Importar archivo CSV.
5. Seleccione el archivo CSV y elija Open (Abrir). Los datos del archivo CSV se agregarán a la tabla.

> **Note**
>
> Si el archivo CSV contiene una o más filas que tienen las mismas claves que los elementos que ya están en la tabla, tendrá la opción de sobrescribir los elementos existentes o agregarlos al final de la tabla. Si elige anexar los elementos, se agregará el sufijo “-Copy” a la clave de cada elemento duplicado para diferenciar los elementos duplicados de los elementos que ya estaban en la tabla.

Visualización de patrones de acceso a datos

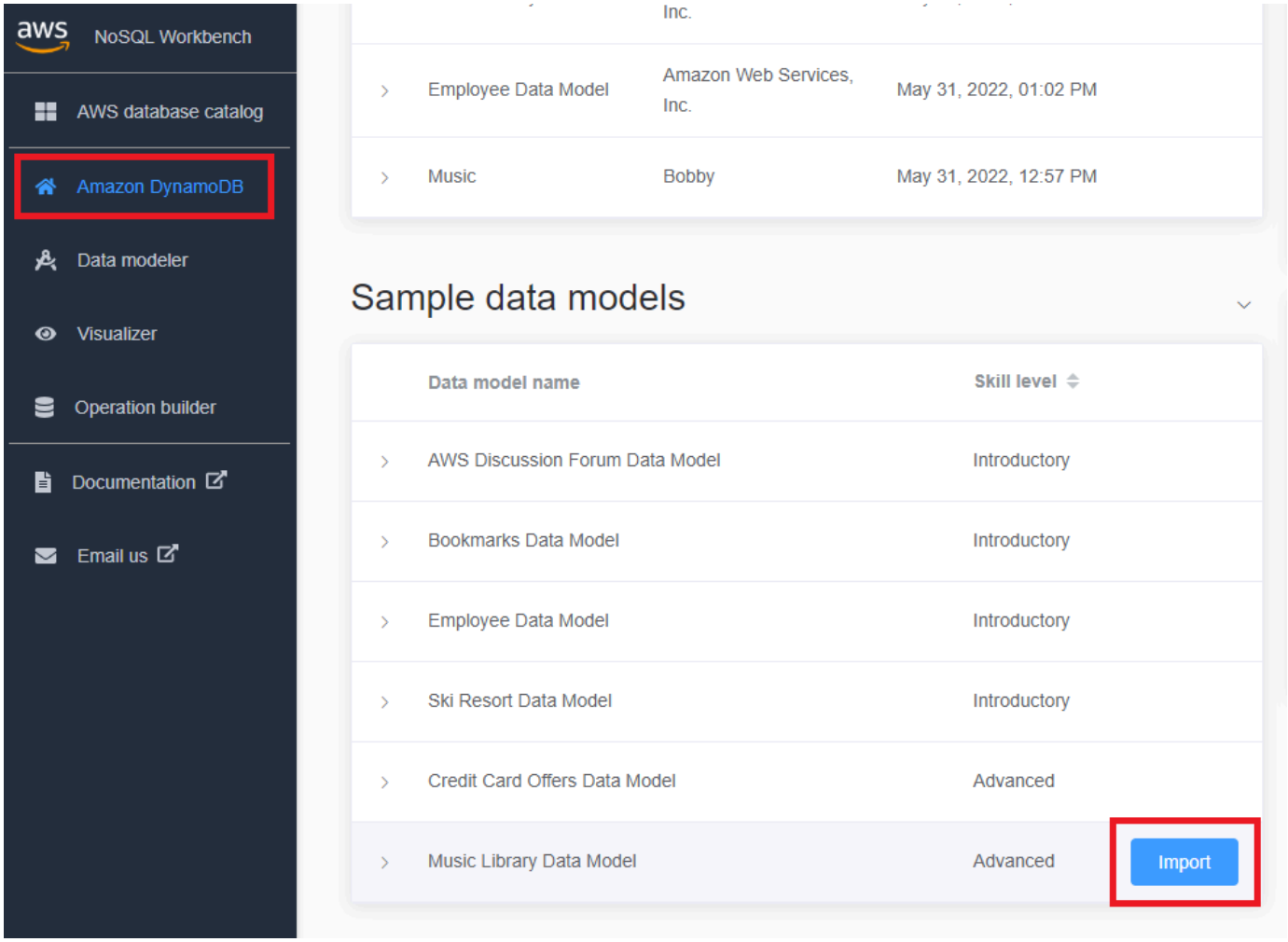
En NoSQL Workbench, las facetas representan los diferentes patrones de acceso a datos de una aplicación para Amazon DynamoDB. Las facetas pueden ayudarle a visualizar su modelo de datos

cuando varios tipos de datos están representados por una clave de clasificación. Las facetas le permiten ver un subconjunto de datos de una tabla, sin tener que ver registros que no cumplen las restricciones de la faceta. Las facetas se consideran una herramienta visual de modelado de datos y no existen como una construcción utilizable en DynamoDB, ya que son puramente una ayuda para modelar patrones de acceso.

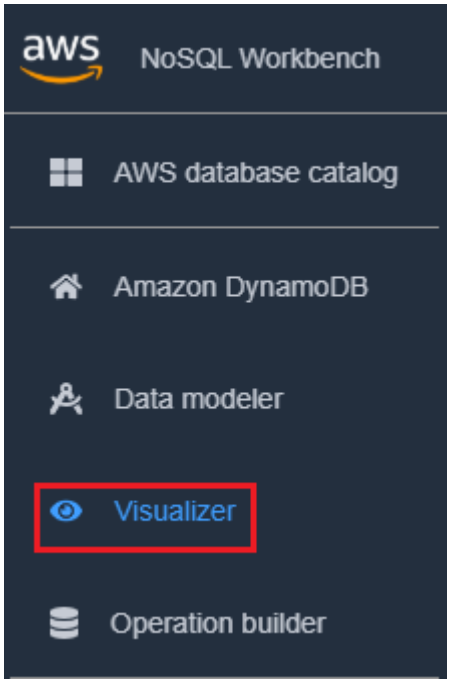
Para ver un ejemplo de facetas, puede importar uno de nuestros modelos de datos de muestra con facetas como parte de la plantilla del modelo de datos.

Importar modelo de datos de muestra

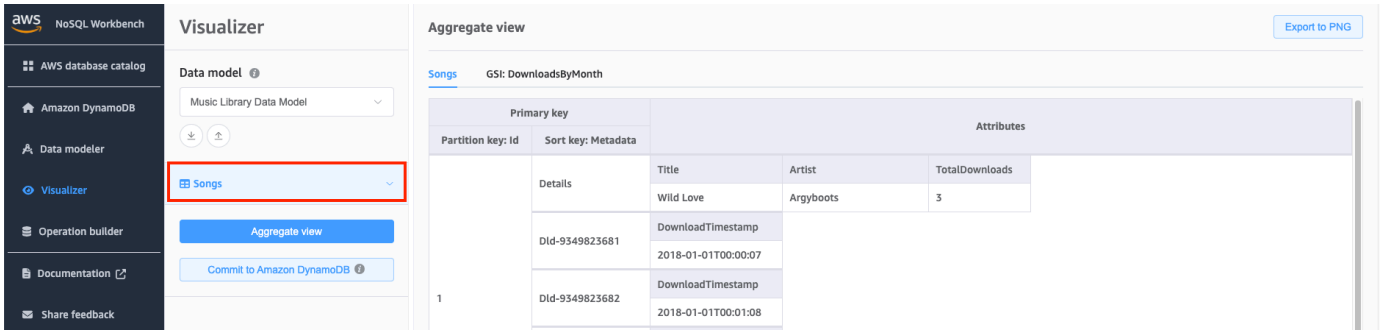
1. En la izquierda, elija Amazon DynamoDB.
2. En la sección de modelos de datos de ejemplo, pase el puntero sobre el modelo de datos de biblioteca de música y elija Import (Importar).



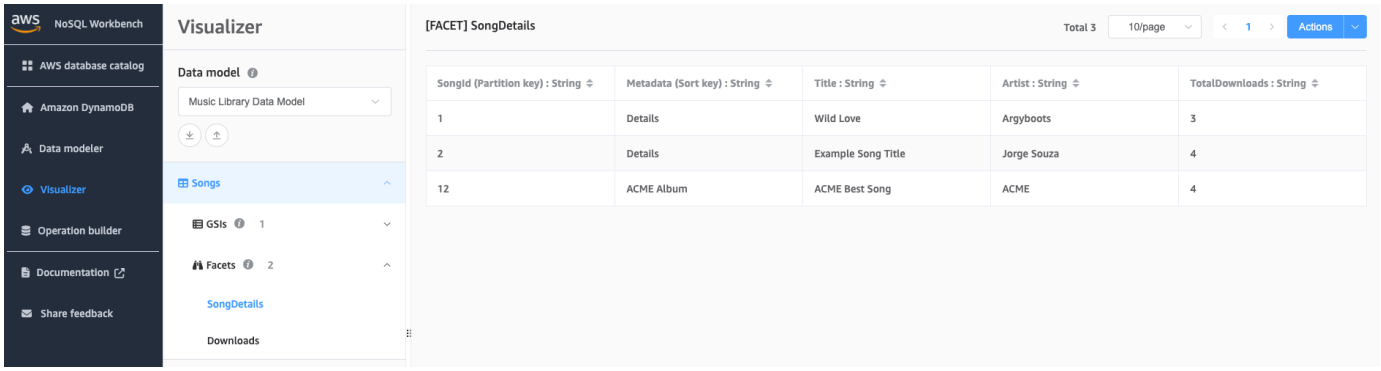
3. En el panel de navegación de la izquierda, elija el icono visualizer (visualizador).



4. Elija la tabla Songs (Canciones) para expandirla. Se mostrará una vista agregada de los datos.



5. Elija la flecha desplegable Facets (Facetas) para expandir las facetas disponibles.
6. Elija la faceta SongDetails para visualizar los datos con dicha faceta aplicada.



También puede editar las definiciones de facetas mediante el modelador de datos. Para obtener más información, consulte Edición de un modelo de datos existente.

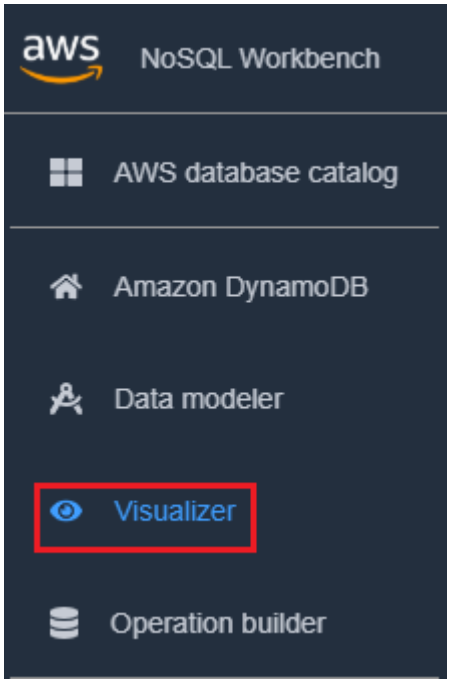
Visualización de todas las tablas de un modelo de datos mediante la vista agregada

La vista agregada en NoSQL Workbench para Amazon DynamoDB representa todas las tablas de un modelo de datos. Para cada tabla, aparece la siguiente información:

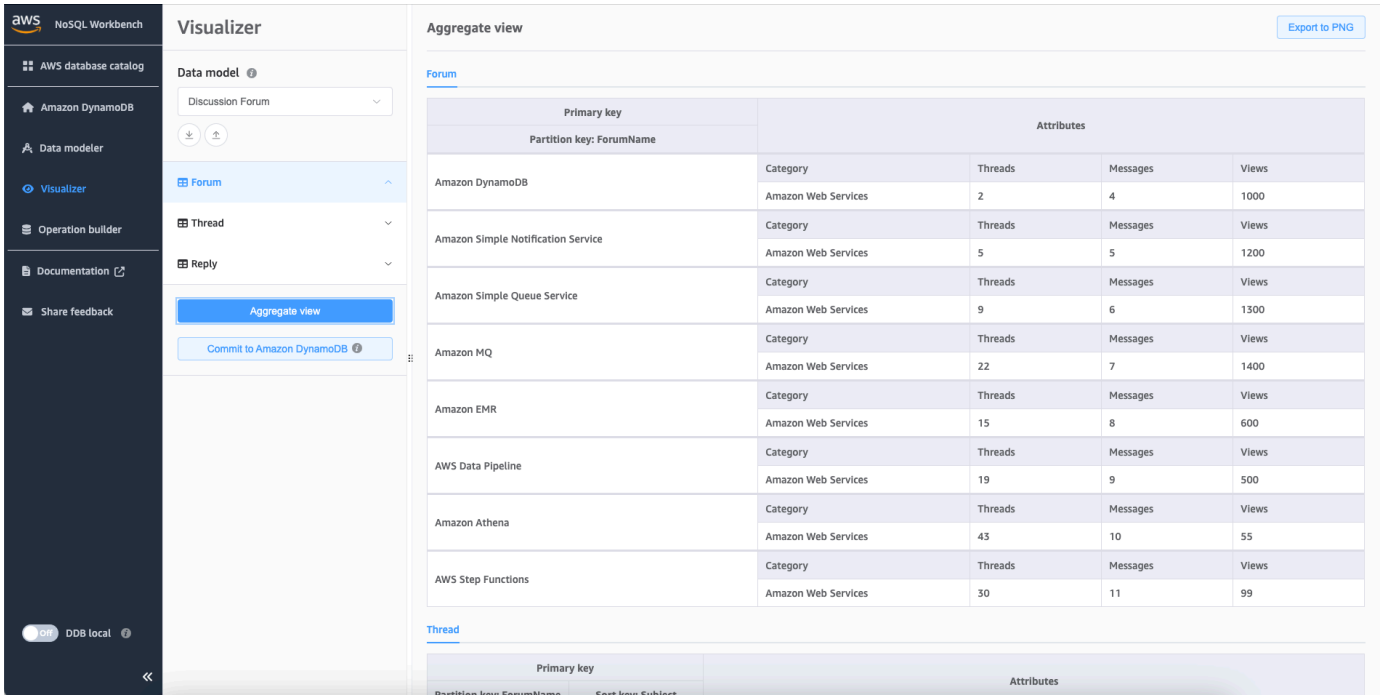
- Nombres de las columnas de la tabla
- Datos de ejemplo
- Todos los índices secundarios globales asociados a la tabla. Para cada índice se muestra la siguiente información:
  - Nombres de las columnas del índice
  - Datos de ejemplo

Para ver toda la información de la tabla

1. En el panel de navegación de la izquierda, elija el icono visualizer (visualizador).



2. En el visualizador, elija Aggregate view (Vista agregada).



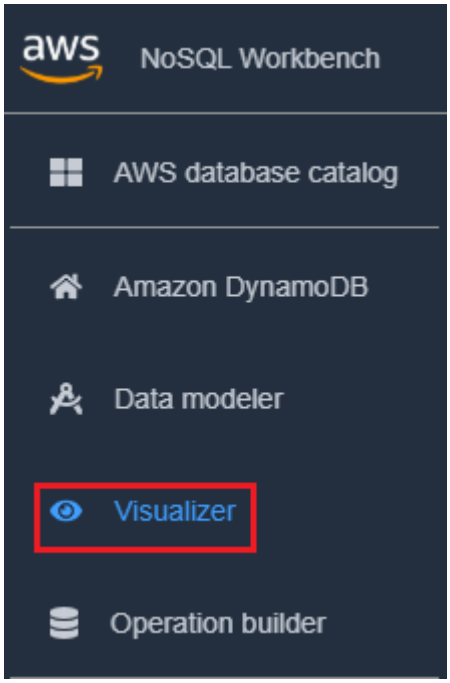
Confirmación de un modelo de datos en DynamoDB

Cuando esté satisfecho con su modelo de datos, puede confirmar el modelo en Amazon DynamoDB.

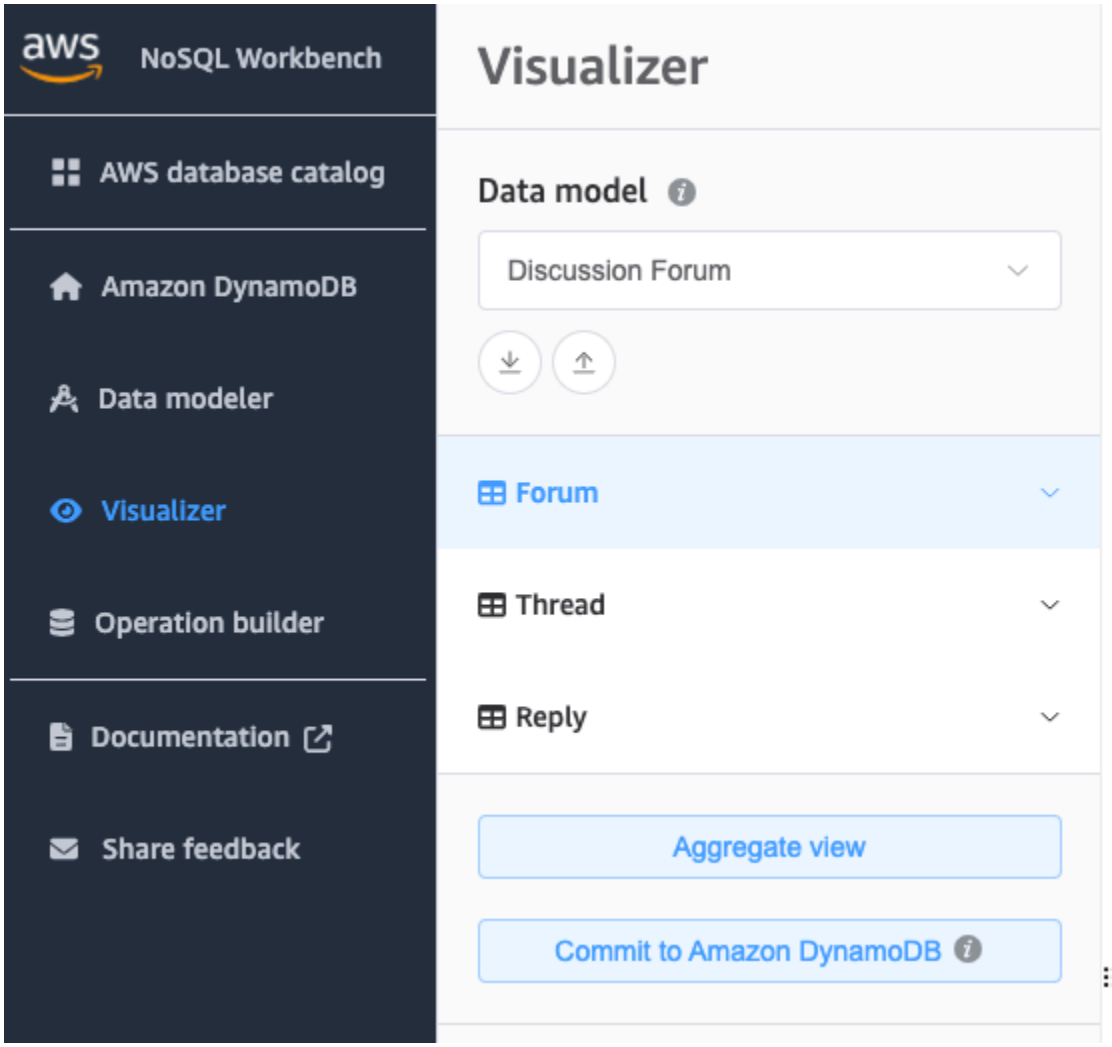
> **Note**
>
> - Esta acción produce la creación de recursos del lado del servidor en AWS para las tablas e índices secundarios globales representados en el modelo de datos.
> - Las tablas se crean con las siguientes características:
>   - El escalado automático se establece en un 70 % del objetivo de utilización.
>   - La capacidad aprovisionada se establece en 5 unidades de capacidad de lectura y en 5 unidades de capacidad de escritura.
> - Los índices secundarios globales se crean con una capacidad aprovisionada de 10 unidades de capacidad de lectura y 5 unidades de capacidad de escritura.

Para confirmar un modelo de datos en DynamoDB

1. En el panel de navegación de la izquierda, elija el icono visualizer (visualizador).



2. Elija Commit to DynamoDB (Confirmar en DynamoDB).



3. Elija una conexión ya existente o cree una nueva; para ello, elija la pestaña Add new remote connection (Añadir nueva conexión remota).

   - Para añadir una nueva conexión, especifique la siguiente información:
     - Account Alias (Alias de cuenta)
     - Región de AWS
     - ID de clave de acceso
     - Clave de acceso secreta

     Para obtener más información sobre cómo obtener las claves de acceso, consulte Obtención de una clave de acceso de AWS.
   - Si lo desea, puede especificar lo siguiente:
     - Token de sesión

- ARN del rol de IAM
- Si no desea registrarse para obtener una cuenta de nivel gratuito y prefiere usar DynamoDB Local (versión descargable):
  1. Elija la pestaña Add a new DynamoDB local connection (Agregar una nueva conexión local de DynamoDB).
  2. Especifique el Connection name (Nombre de conexión) y el Port (Puerto).

4. Elija Commit (Confirmar).

> **Note**
>
> Si ha instalado DynamoDB local como parte de la configuración de NoSQL Workbench, deberá activar DynamoDB local mediante la opción Servidor de DynamoDB local situada en la parte inferior izquierda de la pantalla de NoSQL Workbench. Consulte Instalar NoSQL Workbench para DynamoDB para obtener más información sobre esta opción.

Exploración de conjuntos de datos y creación de operaciones con NoSQL Workbench

NoSQL Workbench para Amazon DynamoDB proporciona una completa interfaz gráfica de usuario para desarrollar y probar consultas. Puede utilizar el generador de operaciones en NoSQL Workbench para ver, explorar y consultar conjuntos de datos en directo. El generador de operaciones estructuradas admite la expresión de proyecciones y de condiciones y, además, genera código de muestra en varios idiomas. Puede clonar tablas directamente de una cuenta de Amazon DynamoDB en otra en distintas regiones. También puede clonar tablas directamente entre DynamoDB local y una cuenta de Amazon DynamoDB para copiar más rápidamente el esquema de claves de la tabla (y, opcionalmente, el esquema y los elementos de GSI) entre sus entornos de desarrollo. Puede guardar hasta 50 operaciones de datos de DynamoDB en el generador de operaciones.

Temas

- Conexión a conjuntos de datos en directo
- Generación de operaciones complejas
- Clonación de tablas con NoSQL Workbench
- Exportación de datos a un archivo CSV

Conexión a conjuntos de datos en directo

Para conectarse a las tablas de Amazon DynamoDB con NoSQL Workbench, primero se debe conectar a la cuenta de AWS.

Para añadir una conexión a su base de datos

1. En NoSQL Workbench, en el panel de navegación de la izquierda, seleccione el icono Operation builder (Generador de operaciones).
2. Elija Agregar conexión.
3. Especifique la siguiente información:

   - Connection name
   - Región de AWS
   - ID de clave de acceso
   - Clave de acceso secreta

   Para obtener más información sobre cómo obtener las claves de acceso, consulte Obtención de una clave de acceso de AWS.

   Si lo desea, puede especificar lo siguiente:

   - Token de sesión
   - ARN del rol de IAM
4. Elija Conectar.

   Si no desea registrarse para obtener una cuenta de nivel gratuito y prefiere usar DynamoDB Local (versión descargable):

   a. Elija la pestaña Local en la pantalla de conexión.
   b. Especifique la siguiente información:

      - Connection name
      - Puerto

   c. Seleccione el botón Connect (Conectar).

> **Note**
>
> Para conectarse a DynamoDB local, inicie DynamoDB local manualmente desde su terminal (consulte implementación de DynamoDB localmente en el equipo) o inicie DynamoDB local directamente mediante el conmutador DDB local del menú de navegación de NoSQL Workbench. Asegúrese de que el puerto de conexión sea el mismo que el puerto local de DynamoDB.

5. En la conexión creada, elija Open (Abrir).

Después de conectarse a su base de datos de DynamoDB, la lista de tablas disponibles aparece en el panel izquierdo. Elija una de las tablas para devolver una muestra de los datos almacenados en la tabla.

Ahora puede ejecutar consultas en la tabla seleccionada.

Para ejecutar consultas en una tabla, consulte la siguiente sección sobre operaciones de creación, consulte Generación de operaciones complejas.

Generación de operaciones complejas

El generador de operaciones en NoSQL Workbench para Amazon DynamoDB proporciona una interfaz visual donde puede realizar operaciones complejas de planos de datos. Incluye compatibilidad con expresiones de proyección y de condición. Una vez que haya creado una operación, puede guardarla para su uso posterior (se pueden guardar hasta 50 operaciones). A continuación, puede examinar una lista de las operaciones de plano de datos utilizadas con frecuencia en la pestaña Operaciones guardadas y utilízalos para rellenar y generar automáticamente una nueva operación. También puede generar el código de muestra para estas operaciones en varios idiomas.

NoSQL Workbench admite la creación de instrucciones PartiQL para DynamoDB, que le permite interactuar con DynamoDB utilizando un lenguaje de consulta compatible con SQL. NoSQL Workbench también admite la creación de operaciones de la API CRUD de DynamoDB.

Para usar NoSQL Workbench para crear operaciones, en el panel de navegación de la izquierda, elija el icono Operation builder (Generador de operaciones).

Temas

- Creación de instrucciones PartiQL
- Generación de operaciones API

Creación de instrucciones PartiQL

Para utilizar NoSQL Workbench para crear instrucciones PartiQL para DynamoDB elija Editor de PartiQL junto a la parte superior de la interfaz de usuario de NoSQL Workbench.

Puede crear los siguientes tipos de instrucciones PartiQL en el generador de operaciones.

Temas

- Instrucciones Singleton
- Transacciones
- Lote

Instrucciones Singleton

Para ejecutar o generar código para una instrucción PartiQL, haga lo siguiente.

1. Elija el editor de PartiQL junto a la parte superior de la ventana.
2. Ingrese una Instrucción PartiQL válida.
3. Si la instrucción utiliza parámetros:
    a. Elegir Parámetros de solicitudes opcionales.
    b. Elegir Agregar parámetro nuevo.
    c. Ingrese el tipo de atributo y el valor.
    d. Si desea añadir parámetros adicionales, repita los pasos b y c.
4. Si desea generar código, seleccione Generate code (Generar código).

    Seleccione el idioma que desee en las pestañas mostradas. Ahora puede copiar este código y utilizarlo en su aplicación.
5. Si desea que la operación se ejecute inmediatamente, seleccione Execute (Ejecutar).
6. Si desea guardar esta operación para su uso posterior, elija Save operation (Guardar operación). A continuación, escriba un nombre para la operación y elija Save (Guardar).

Transacciones

Para ejecutar o generar el código para una transacción PartiQL, haga lo siguiente.

1. Elija PartiQLTransaction del menú desplegable Más operaciones.
2. Elegir Agregar una nueva instrucción.
3. Ingrese una Instruccción PartiQL válida.

> **Note**
>
> Las operaciones de lectura y escritura no se admiten en la misma solicitud de transacción PartiQL. Una instrucción SELECT no puede estar en la misma solicitud con las instrucciones INSERT, UPDATE y DELETE. Consulte Realización de transacciones con PartiQL para DynamoDB para obtener más información.

4. Si su instrucción utiliza parámetros
   a. Elegir Parámetros de solicitudes opcionales.
   b. Elegir Agregar parámetro nuevo.
   c. Ingrese el tipo de atributo y el valor.
   d. Si desea añadir parámetros adicionales, repita los pasos b y c.
5. Si desea agregar más instrucciones, repita los pasos 2 a 4.
6. Si desea generar código, seleccione Generate code (Generar código).

   Seleccione el idioma que desee en las pestañas mostradas. Ahora puede copiar este código y utilizarlo en su aplicación.
7. Si desea que la operación se ejecute inmediatamente, seleccione Execute (Ejecutar).
8. Si desea guardar esta operación para su uso posterior, elija Save operation (Guardar operación). A continuación, escriba un nombre para la operación y elija Save (Guardar).

Lote

Para ejecutar o generar código para un lote de PartiQL, haga lo siguiente.

1. Elija PartiQLBatch del menú desplegable Más operaciones.
2. Elegir Agregar una nueva instrucción.

3. Ingrese una Instrucción PartiQL válida.

> **Note**
>
> Las operaciones de lectura y escritura no se admiten en la misma solicitud por lotes PartiQL, lo que significa que una instrucción SELECT no puede estar en la misma solicitud con instrucciones INSERT, UPDATE y DELETE. No se permiten operaciones de escritura en el mismo elemento. Al igual que con la operación BatchGetItem, solo se admiten operaciones de lectura singleton. No se admiten las operaciones de análisis y consulta. Consulte Ejecución de operaciones por lote con PartiQL para DynamoDB para obtener más información.

4. Si la instrucción utiliza parámetros:

   a. Elegir Parámetros de solicitudes opcionales.

   b. Elegir Agregar parámetro nuevo.

   c. Ingrese el tipo de atributo y el valor.

   d. Si desea añadir parámetros adicionales, repita los pasos b y c.

5. Si desea agregar más instrucciones, repita los pasos 2 a 4.

6. Si desea generar código, seleccione Generate code (Generar código).

   Seleccione el idioma que desee en las pestañas mostradas. Ahora puede copiar este código y utilizarlo en su aplicación.

7. Si desea que la operación se ejecute inmediatamente, seleccione Execute (Ejecutar).

8. Si desea guardar esta operación para su uso posterior, elija Save operation (Guardar operación). A continuación, escriba un nombre para la operación y elija Save (Guardar).

Generación de operaciones API

Para utilizar NoSQL Workbench para crear API CRUD de DynamoDB, seleccione Creador de operaciones de la izquierda de la interfaz de usuario de NoSQL Workbench.

A continuación, seleccione Abierto y elija una conexión.

Puede realizar las siguientes operaciones en el generador de operaciones.

- Eliminar tabla

- Crear tabla
- Actualizar tabla

- Poner elemento
- Actualizar elemento
- Eliminar elemento
- Query
- Scan
- Transacción de obtención de elementos
- Transacción de escritura de elementos

Eliminar tabla

Para ejecutar una operación `Delete Table`, siga los pasos que se muestran a continuación.

1. Encuentre la tabla que desea eliminar en la sección Tablas.
2. Seleccione Eliminación de tabla en el menú de puntos suspensivos horizontales.
3. Confirme que desea eliminar la tabla ingresando el Nombre de la tabla.
4. Seleccione Eliminar.

Para obtener más información acerca de esta operación, consulte Eliminar tabla en la Referencia de la API de Amazon DynamoDB.

Eliminación de GSI

Para ejecutar una operación `Delete GSI`, siga los pasos que se muestran a continuación.

1. Encuentre el GSI de una tabla que desea eliminar de la sección Tablas.
2. Seleccione Eliminación de GSI en el menú de puntos suspensivos horizontales.
3. Confirme que desea eliminar GSI ingresando el Nombre de GSI.
4. Seleccione Eliminar.

Para obtener más información acerca de esta operación, consulte Eliminar tabla en la Referencia de la API de Amazon DynamoDB.

Crear tablas

Para ejecutar una operación `Create Table`, siga los pasos que se muestran a continuación.

1. Elija el icono + junto a la sección Tablas.
2. Ingrese el nombre de tabla que desee.
3. Cree una clave de partición.
4. Opcional: cree una clave de clasificación.
5. Para personalizar la configuración de capacidad y desmarcar la casilla que hay junto a Uso de la configuración de capacidad predeterminada.
   - Ahora puede seleccionar Provisioned (Aprovisionada) o On-demand capacity (Capacidad bajo demanda).

     Si se selecciona Aprovisionada, puede establecer el mínimo y el máximo de las unidades de capacidad de lectura y escritura. También puede habilitar o deshabilitar el escalado automático.
   - Si la tabla está configurada actualmente en Bajo demanda, no podrá especificar un rendimiento aprovisionado.
   - Si cambia el rendimiento de Bajo demanda a Aprovisionado, el escalado automático se aplicará automáticamente a todos los GSI con mínimo: 1, máximo: 10; destino: 70 %.
6. Seleccione Omisión de GSI y creación para crear esta tabla sin un GSI. Si lo desea, puede seleccionar Siguiente para crear un GSI con esta nueva tabla.

Para obtener más información acerca de esta operación, consulte Crear tabla en la Referencia de la API de Amazon DynamoDB.

Creación de GSI

Para ejecutar una operación `Create GSI`, siga los pasos que se muestran a continuación.

1. Encuentre una tabla a la que quiera agregar un GSI.
2. En el menú de puntos suspensivos horizontales, seleccione Creación de GSI.
3. Asigne un nombre al GSI en Nombre de índice.
4. Cree una clave de partición.
5. Opcional: cree una clave de clasificación.

6. Elija una opción de tipo de proyección en el menú desplegable.
7. Seleccione Creación de GSI.

Para obtener más información acerca de esta operación, consulte Crear tabla en la Referencia de la API de Amazon DynamoDB.

Actualizar tabla

Para actualizar la configuración de capacidad de una tabla con una operación `Update Table`, haga lo siguiente.

1. Encuentre la tabla para la que desea actualizar la configuración de capacidad.
2. En el menú de puntos suspensivos horizontales, seleccione Actualización de la configuración de capacidad.
3. Seleccione Aprovisionada o capacidad Bajo demanda.

   Si se selecciona Aprovisionada, puede establecer el mínimo y el máximo de las unidades de capacidad de lectura y escritura. También puede habilitar o deshabilitar el escalado automático.
4. Seleccione Update (Actualizar).

Para obtener más información acerca de esta operación, consulte Actualizar tabla en la Referencia de la API de Amazon DynamoDB.

Actualización de GSI

Para actualizar la configuración de capacidad de GSI con una operación `Update Table`, haga lo siguiente.

> **Note**
>
> De forma predeterminada, los índices secundarios globales heredan la configuración de capacidad de la tabla base. Los índices secundarios globales pueden tener un modo de capacidad diferente solo cuando la tabla base está en el modo de capacidad aprovisionada. Al crear un índice secundario global en una tabla en modo aprovisionado, debe especificar las unidades de capacidad de lectura y escritura para la carga de trabajo prevista de ese índice. Para obtener más información, consulte Consideraciones sobre el rendimiento aprovisionado para los índices secundarios globales.

1. Encuentre el GSI para el que desea actualizar la configuración de capacidad.
2. En el menú de puntos suspensivos horizontales, seleccione Actualización de la configuración de capacidad.
3. Ahora puede seleccionar Provisioned (Aprovisionada) o On-demand capacity (Capacidad bajo demanda).

   Si se selecciona Aprovisionada, puede establecer el mínimo y el máximo de las unidades de capacidad de lectura y escritura. También puede habilitar o deshabilitar el escalado automático.
4. Seleccione Update (Actualizar).

Para obtener más información acerca de esta operación, consulte Actualizar tabla en la Referencia de la API de Amazon DynamoDB.

Poner elemento

Se crea un elemento mediante la operación `Put Item`. Para ejecutar o generar código para una operación `Put Item`, haga lo siguiente.

1. Encuentre la tabla en la que desea crear un elemento.
2. En el menú desplegable de Acciones, seleccione Creación de elemento.
3. Escriba el valor de la clave de partición.
4. Escriba el valor de la clave de clasificación, si hay una.
5. Si desea añadir atributos que no sean de clave, haga lo siguiente:

   a. Seleccione + Agregación de otros atributos.

   b. Especifique los valores de Attribute name (Nombre de atributo), Type (Tipo) y Value (Valor).
6. Si debe satisfacerse una expresión de condición para que la operación `Put Item` se realice correctamente, haga lo siguiente:

   a. Elija Condition (Condición).

   b. Especifique el nombre de atributo, el operador de comparación, el tipo de atributo y el valor de atributo.

   c. Si se necesitan otras condiciones, seleccione Condition (Condición) de nuevo.

   Para obtener más información, consulte Ejemplo de la CLI de expresión de condición de DynamoDB.

7. Si desea generar código, seleccione Generate code (Generar código).

   Seleccione el idioma que desee en las pestañas mostradas. Ahora puede copiar este código y utilizarlo en su aplicación.
8. Si desea que la operación se ejecute inmediatamente, seleccione Execute (Ejecutar).
9. Si desea guardar esta operación para su uso posterior, elija Save operation (Guardar la operación), escriba el nombre de la operación y elija Save (Guardar).

Para obtener más información acerca de esta operación, consulte PutItem en la Referencia de la API de Amazon DynamoDB.

Actualizar elemento

Para ejecutar o generar código para una operación `Update Item`, haga lo siguiente:

1. Encuentre la tabla en la que desea actualizar un elemento.
2. Seleccione el elemento.
3. Escriba el nombre y el valor del atributo para la expresión seleccionada.
4. Si desea agregar más expresiones, elija otra de la lista desplegable Actualización de expresión y, a continuación, seleccione el icono +.
5. Si debe satisfacerse una expresión de condición para que la operación `Update Item` se realice correctamente, haga lo siguiente:

   a. Elija Condition (Condición).

   b. Especifique el nombre de atributo, el operador de comparación, el tipo de atributo y el valor de atributo.

   c. Si se necesitan otras condiciones, seleccione Condition (Condición) de nuevo.

   Para obtener más información, consulte Ejemplo de la CLI de expresión de condición de DynamoDB.
6. Si desea generar código, seleccione Generate code (Generar código).

   Elija la pestaña del idioma que desee. Ahora puede copiar este código y utilizarlo en su aplicación.
7. Si desea que la operación se ejecute inmediatamente, seleccione Execute (Ejecutar).

8. Si desea guardar esta operación para su uso posterior, elija Save operation (Guardar la operación), escriba el nombre de la operación y elija Save (Guardar).

Para obtener más información acerca de esta operación, consulte UpdateItem en la Referencia de la API de Amazon DynamoDB.

Eliminar elemento

Para ejecutar una operación `Delete Item`, siga los pasos que se muestran a continuación.

1. Encuentre la tabla de la que desea eliminar un elemento.
2. Seleccione el elemento.
3. Del menú desplegable Acciones, seleccione Eliminación de elemento.
4. Confirme que desea eliminar el elemento seleccionando Eliminación.

Para obtener más información acerca de esta operación, consulte DeleteItem en la Referencia de la API de Amazon DynamoDB.

Duplicado de elemento

Puede duplicar un elemento creando uno nuevo con los mismos atributos. Para duplicar un elemento, haga lo siguiente.

1. Encuentre la tabla en la que desea duplicar un elemento.
2. Seleccione el elemento.
3. Del menú desplegable Acciones, seleccione Duplicación de elemento.
4. Especifique una nueva clave de partición.
5. Especifique una nueva clave de clasificación (si es necesario).
6. Seleccione Ejecución.

Para obtener más información acerca de esta operación, consulte DeleteItem en la Referencia de la API de Amazon DynamoDB.

Consultar

Para ejecutar o generar código para una operación `Query`, haga lo siguiente.

1. Seleccione Consulta en la parte superior de la interfaz de usuario de NoSQL Workbench.
2. Especifique el valor de la clave de partición.
3. Si se necesita una clave de clasificación para la operación `Query`:
   a. Seleccione Sort key (Clave de ordenación).
   b. Especifique el operador de comparación y el valor de atributo.
4. Seleccione Consulta para ejecutar esta operación de consulta. Si necesita más opciones, compruebe la casilla de verificación Más opciones y siga con los pasos siguientes.
5. Si no se deben devolver todos los atributos con el resultado de la operación, seleccione Projection expression (Expresión de proyección).
6. Elija el icono +.
7. Escriba el atributo que se devolverá con el resultado de la consulta.
8. Si se necesitan más atributos, seleccione +.
9. Si debe satisfacerse una expresión de condición para que la operación `Query` se realice correctamente, haga lo siguiente:
   a. Elija Condition (Condición).
   b. Especifique el nombre de atributo, el operador de comparación, el tipo de atributo y el valor de atributo.
   c. Si se necesitan otras condiciones, seleccione Condition (Condición) de nuevo.

   Para obtener más información, consulte Ejemplo de la CLI de expresión de condición de DynamoDB.
10. Si desea generar código, seleccione Generate code (Generar código).

    Elija la pestaña del idioma que desee. Ahora puede copiar este código y utilizarlo en su aplicación.
11. Si desea que la operación se ejecute inmediatamente, seleccione Execute (Ejecutar).
12. Si desea guardar esta operación para su uso posterior, elija Save operation (Guardar la operación), escriba el nombre de la operación y elija Save (Guardar).

Para obtener más información acerca de esta operación, consulte Consulta en la Referencia de la API de Amazon DynamoDB.

Examen

Para ejecutar o generar código para una operación `Scan`, haga lo siguiente.

1. Seleccione Escaneo en la parte superior de la interfaz de usuario de NoSQL Workbench.
2. Seleccione el botón Escanear para realizar esta operación de escaneo básica. Si necesita más opciones, compruebe la casilla de verificación Más opciones y siga con los pasos siguientes.
3. Especifique un nombre de atributo para filtrar los resultados del escaneo.
4. Si no se deben devolver todos los atributos con el resultado de la operación, seleccione Projection expression (Expresión de proyección).
5. Si debe satisfacerse una expresión de condición para que la operación de examen se realice correctamente, haga lo siguiente:

   a. Elija Condition (Condición).
   b. Especifique el nombre de atributo, el operador de comparación, el tipo de atributo y el valor de atributo.
   c. Si se necesitan otras condiciones, seleccione Condition (Condición) de nuevo.

   Para obtener más información, consulte Ejemplo de la CLI de expresión de condición de DynamoDB.
6. Si desea generar código, seleccione Generate code (Generar código).

   Elija la pestaña del idioma que desee. Ahora puede copiar este código y utilizarlo en su aplicación.
7. Si desea que la operación se ejecute inmediatamente, seleccione Execute (Ejecutar).
8. Si desea guardar esta operación para su uso posterior, elija Save operation (Guardar la operación), escriba el nombre de la operación y elija Save (Guardar).

TransactGetItems

Para ejecutar o generar código para una operación `TransactGetItems`, haga lo siguiente.

1. Del menú desplegable Más operaciones en la parte superior de la interfaz de usuario de NoSQL Workbench, elija TransactGetItems.
2. Elija el icono + junto a TransactGetItem.
3. Especifique una clave de partición.

4. Especifique una clave de clasificación (si es necesario).
5. Seleccione Ejecución para realizar la operación, Guardado de operación para guardarla o Generación de código para generar código para ella.

Para obtener más información sobre las transacciones, consulte Transacciones de Amazon DynamoDB.

TransactWriteItems

Para ejecutar o generar código para una operación `TransactWriteItems`, haga lo siguiente.

1. Del menú desplegable Más operaciones en la parte superior de la interfaz de usuario de NoSQL Workbench, elija TransactWriteItems.
2. Elija una operación del menú desplegable Acciones.
3. Elija el icono + cerca de TransactWriteItem.
4. En el menú desplegable Actions, elija la operación que desea realizar.

   - Para `DeleteItem`, siga las instrucciones de la operación Eliminar elemento.
   - Para `PutItem`, siga las instrucciones de la operación Poner elemento.
   - Para `UpdateItem`, siga las instrucciones de la operación Actualizar elemento.

   Para cambiar el orden de las acciones, seleccione una acción en la lista de la izquierda y, a continuación, elija las flechas hacia arriba o hacia abajo para subirla o bajarla en la lista.

   Para eliminar una acción, seleccione la acción de la lista y, a continuación, el icono Delete (Eliminar) (papelera).
5. Seleccione Ejecución para realizar la operación, Guardado de operación para guardarla o Generación de código para generar código para ella.

Para obtener más información sobre las transacciones, consulte Transacciones de Amazon DynamoDB.

Clonación de tablas con NoSQL Workbench

Al clonar tablas, se copiará el esquema de claves de una tabla (y, opcionalmente, el esquema y los elementos de GSI) entre los entornos de desarrollo. Puede clonar una tabla entre DynamoDB local

en una cuenta de Amazon DynamoDB e incluso clonar una tabla de una cuenta en otra en distintas regiones para acelerar la experiencia.

Cancelar una tabla

1. En Generador de operaciones, seleccione su conexión y su región (la selección de región no está disponible para DynamoDB local).
2. Cuando esté conectado a DynamoDB, examine las tablas y seleccione la tabla que desee clonar.
3. En el menú de puntos suspensivos horizontales, seleccione la opción Clonación.
4. Especifique la información del destino del clon:
    a. Seleccione una conexión.
    b. Seleccione una región (la región no está disponible para DynamoDB local).
    c. Escriba un nombre de tabla nuevo.
    d. Elija una opción de clonación:
        i. La opción Esquema de claves está seleccionada de forma predeterminada y no se puede desmarcar. De forma predeterminada, al clonar una tabla, se copiarán la clave principal y la clave de clasificación, si están disponibles.
        ii. La opción Esquema de GSI se selecciona de forma predeterminada si la tabla que se va a clonar tiene un GSI. Al clonar una tabla, se copiarán la clave principal y la clave de clasificación del GSI, si están disponibles. Tiene la opción de desmarcar el esquema de GSI para impedir que se clone. Al clonar una tabla, la configuración de capacidad de la tabla base se copiará como la configuración de capacidad del GSI. Puede utilizar la operación `UpdateTable` del Generador de operaciones para actualizar la configuración de capacidad del GSI de la tabla una vez finalizada la clonación.
5. Introduzca el número de elementos que se debe clonar. Para clonar solo el esquema de claves y, opcionalmente, el esquema de GSI, puede mantener el valor de Elementos que se van a clonar en 0. El número máximo de elementos que puede clonar es 5000.
6. Elija un modo de capacidad:
    a. El modo bajo demanda está seleccionado de forma predeterminada. DynamoDB bajo demanda ofrece precios de pago por solicitud para las solicitudes de lectura y escritura. De este modo, únicamente tendrá que pagar por aquello que utilice. Para obtener más información, consulte Modo bajo demanda de DynamoDB.

b. El modo aprovisionado le permite especificar el número de lecturas y escrituras por segundo que necesita para la aplicación. Puede utilizar el escalado automático para ajustar automáticamente la capacidad aprovisionada de la tabla en respuesta a los patrones de tráfico. Para obtener más información, consulte Modo aprovisionado de DynamoDB.

7. Seleccione Clonar para empezar a clonar.
8. El proceso de clonación se ejecutará en segundo plano. La pestaña Generador de operaciones mostrará una notificación cuando se produzca un cambio en el estado de la tabla de clonación. Para acceder a este estado, seleccione la pestaña Generador de operaciones y, a continuación, el botón de flecha. El botón de flecha se encuentra en el widget de estado de la tabla de clonación, ubicado cerca de la parte inferior de la barra lateral del menú.

Exportación de datos a un archivo CSV

Puede exportar los resultados de una consulta desde el Generador de operaciones a un archivo CSV. Esto le permite cargar los datos en una hoja de cálculo o procesarlos utilizando el lenguaje de programación preferido.

Exportación a CSV

1. En el Generador de operaciones, ejecute una operación de su elección, como un análisis o una consulta.

> **Note**
>
> - Solo puede exportar los resultados de las operaciones de lectura de la API y de las instrucciones PartiQL a un archivo CSV. No puede exportar los resultados de las instrucciones de lectura de transacciones.
> - Actualmente, puede exportar los resultados página por página a un archivo CSV. Si hay varias páginas de resultados, debe exportar cada página individualmente.

2. Seleccione los elementos que desea exportar de los resultados.
3. En el menú desplegable Acciones, elija Exportación como CSV.
4. Elija un nombre de archivo y una ubicación para el archivo CSV y seleccione Save (Guardar).

Modelos de datos de ejemplo para NoSQL Workbench

La página principal del modelador y el visualizador muestran una serie de modelos de ejemplo que se incluyen con NoSQL Workbench. En esta sección se describen estos modelos y sus usos potenciales.

Temas

- Modelo de datos de empleados
- Modelo de datos del foro de debate
- Modelo de datos de biblioteca de música
- Modelo de datos de la estación de esquí
- Modelo de datos de ofertas de tarjetas de crédito
- Modelo de datos de marcadores

Modelo de datos de empleados

Este modelo de datos es un modelo introductorio. Representa los detalles básicos de un empleado, como un alias único, nombre, apellido, designación, gerente y habilidades.

Este modelo de datos representa algunas técnicas como el manejo de atributos complejos, como tener más de una habilidad. Este modelo es también un ejemplo de relación de uno a varios a través del gerente y sus empleados subordinados que se ha logrado mediante el índice secundario DirectReports.

Los patrones de acceso que este modelo de datos facilita son:

- Recuperación de un registro de empleado utilizando el alias de inicio de sesión del empleado, facilitada por una tabla llamada `Employee`.
- Buscar empleados por nombre, facilitado por el índice secundario global de la tabla Empleado llamado `Name`.
- Recuperación de todos los informes directos de un gerente utilizando el alias de inicio de sesión del gerente, facilitada por el índice secundario global de la tabla Empleado llamado `DirectReports`.

Modelo de datos del foro de debate

Este modelo de datos representa un foro de debate. Con este modelo los clientes pueden interaccionar con la comunidad de desarrolladores, plantear preguntas y contestar a las publicaciones de otros clientes. Cada servicio de AWS tiene un foro específico. Cualquier persona puede iniciar un nuevo hilo de debate publicando un mensaje en un foro, y cada hilo recibe un cierto número de respuestas.

Los patrones de acceso que este modelo de datos facilita son:

- Recuperación de un registro de foro utilizando el nombre del foro, facilitada por una tabla llamada `Forum`.
- Recuperación de un hilo específico o todos los hilos de un foro, facilitada por una tabla llamada `Thread`.
- Busque respuestas utilizando la dirección de correo electrónico del usuario de publicación, facilitado por el índice secundario global de la tabla Respuesta llamado `PostedBy-Message-Index`.

Modelo de datos de biblioteca de música

Este modelo de datos representa una biblioteca de música que tiene una gran colección de canciones y muestra sus canciones más descargadas casi en tiempo real.

Los patrones de acceso que este modelo de datos facilita son:

- Recuperación de un disco de canción, facilitada por una tabla llamada `Songs`.
- Recuperación de un registro de descarga específico o todos los registros de descarga de una canción, facilitada por una tabla llamada `Songs`.
- Recuperación de un registro de recuento mensual específico de descargas o todos los registros de recuento de descargas mensuales de una canción, facilitada por una tabla llamada `Song`.
- Recuperación de todos los registros (incluidos los registros de canciones, los registros de descargas y los registros de recuento de descargas mensuales) de una canción, facilitada por una tabla llamada `Songs`.
- Buscar la mayoría de las canciones descargadas, facilitado por el índice secundario global de la tabla Canciones llamado `DownloadsByMonth`.

Modelo de datos de la estación de esquí

Este modelo de datos representa una estación de esquí que tiene una gran colección de datos para cada telesilla recopilada diariamente.

Los patrones de acceso que este modelo de datos facilita son:

- Recuperación de todos los datos de un remonte determinado o complejo general, dinámico y estático, facilitada por una tabla llamada `SkiLifts`.
- Recuperación de todos los datos dinámicos (incluidos los transportes únicos, la cobertura de nieve, el peligro de avalancha y el estado del ascensor) para un telesilla o el centro vacacional en una fecha específica, facilitada por una tabla llamada `SkiLifts`.
- Recuperación de todos los datos estáticos (incluyendo si el transporte es solo para experimentados, pies verticales que el transporte se eleva y tiempo de conducción en el transporte) para un telesilla específico, facilitado por una mesa llamada `SkiLifts`.
- Recuperación de la fecha de los datos registrados para un remonte específico o el complejo general ordenado por usuarios individuales totales, facilitada por el índice secundario global de la tabla SkiLifts llamado `SkiLiftsByRiders`.

Modelo de datos de ofertas de tarjetas de crédito

Este modelo de datos es utilizado por una aplicación de ofertas de tarjetas de crédito.

Un proveedor de tarjeta de crédito produce ofertas a lo largo del tiempo. Estas ofertas incluyen transferencias de saldo sin cargos, mayores límites de crédito, tasas de interés más bajas, devolución de efectivo y millas aéreas. Después de que un cliente acepte o rechace estas ofertas, el estado de la oferta correspondiente se actualiza en consecuencia.

Los patrones de acceso que este modelo de datos facilita son:

- Recuperación de registros de cuenta usando `AccountId`, según lo facilitado en la tabla principal.
- Recuperación de todas las cuentas con pocos elementos previstos, facilitada por el índice secundario `AccountIndex`.
- Recuperación de cuentas y todos los registros de oferta asociados con esas cuentas mediante el uso de `AccountId`, facilitado por la tabla principal.
- Recuperación de cuentas y registros de ofertas específicas asociadas con esas cuentas mediante el uso de `AccountId` y `OfferId`, según lo facilitado en la tabla principal.

- Recuperación de todos los registros de oferta `ACCEPTED/DECLINED` de `OfferType` específicos asociados con cuentas usando `AccountId`, `OfferType`, y `Status`, según lo facilitado por el índice secundario `GSI1`.
- Recuperación de ofertas y registros de elementos de oferta asociados utilizando `OfferId`, según lo facilitado en la tabla principal.

Modelo de datos de marcadores

Este modelo de datos se utiliza marcadores de tienda para los clientes.

Un cliente puede tener muchos marcadores y un marcador puede pertenecer a muchos clientes. Este modelo de datos representa una relación de muchos a muchos.

Los patrones de acceso que este modelo de datos facilita son:

- Una sola consulta de `customerId` ahora puede devolver datos de clientes, así como marcadores.
- Un índice `ByEmail` de consulta devuelve los datos del cliente por dirección de correo electrónico. Tenga en cuenta que este índice no recupera los marcadores.
- Un índice de consulta `ByUrl` obtiene datos de marcadores por URL. Tenga en cuenta que tenemos customerId como clave de ordenación para el índice porque la misma URL puede ser reservada por varios clientes.
- Un índice de consulta `ByCustomerFolder` obtiene marcadores por carpeta para cada cliente.

Historial de versiones de NoSQL Workbench

En la tabla siguiente se describen los cambios importantes en cada versión de la herramienta cliente de NoSQL Workbench.

Versión	Cambio	Descripción	Fecha
3.13.5	El modo de capacidad para la configuración de tabla predeterminada ahora está disponible bajo demanda	Al crear una tabla con la configuración predeterminada, DynamoDB crea una tabla que usa el modo de capacidad bajo	24 de febrero de 2025

Versión	Cambio	Descripción	Fecha
		demanda en lugar del modo de capacidad aprovisionada.	
3.13.0	Mejoras en el generador de operaciones de NoSQL Workbench	NoSQL Workbench ahora incluye compatibilidad nativa para el modo oscuro. Se han mejorado las operaciones de tablas y elementos en el generador de operaciones. Los resultados de los elementos y la información de solicitud del generador de operaciones están disponibles en formato JSON.	24 de abril de 2024

Versión	Cambio	Descripción	Fecha
3.12.0	Clonar tablas con NoSQL Workbench y devolver la capacidad consumida	Ahora puede clonar tablas entre DynamoDB local y una cuenta de servicio web de DynamoDB o entre cuentas de servicio web de DynamoDB para acelerar las iteraciones de desarrollo. Observa la cantidad de RCU o WCU consumidas después de ejecutar una operación con el Generador de operaciones. Hemos soluciona do el problema de anulación de datos al importar desde un archivo CSV.	26 de febrero de 2024

Versión	Cambio	Descripción	Fecha
3.11.0	Mejoras de DynamoDB local	Ahora puede especificar el puerto al lanzar la instancia de DynamoDB local integrada. NoSQL Workbench ahora se puede instalar en Windows sin derechos de administrador. Hemos actualizado las plantillas del modelo de datos.	17 de enero de 2024
3.10.0	Compatibilidad nativa con Apple Silicon	NoSQL Workbench ahora incluye compatibilidad nativa para Mac con Apple Silicon. Ahora puede configurar un formato de generación de datos de ejemplo para los atributos del tipo `Number`.	5 de diciembre de 2023
3.9.0	Mejoras en el modelador de datos	El visualizador ahora admite la confirmación de modelos de datos en DynamoDB local con la opción de sobrescribir las tablas existentes.	3 de noviembre de 2023

Versión	Cambio	Descripción	Fecha
3.8.0	Generación de datos de muestra	NoSQL Workbench ahora admite la generación de datos de muestra para sus modelos de datos de DynamoDB.	25 de septiembre de 2023
3.6.0	Mejoras en el generador de operaciones	Mejoras en la administración de las conexiones en el generador de operaciones. Los nombres de los atributos en Modelador de datos pueden cambiarse ahora sin eliminar datos. Otras correcciones de errores.	11 de abril de 2023
3.5.0	Compatibilidad con nuevas regiones de AWS	NoSQL Workbench ahora es compatible con las regiones ap-south-2, ap-southeast-3, ap-southeast-4, eu-central-2, eu-south-2, me-central-1 y me-west-1.	23 de febrero de 2023

Versión	Cambio	Descripción	Fecha
3.4.0	Compatibilidad con DynamoDB local	NoSQL Workbench admite ahora la instalación de DynamoDB local como parte del proceso de instalación.	6 de diciembre de 2022
3.3.0	Asistencia para operaciones del plano de control	Operation Builder admite ahora operaciones de plano de control.	1 de junio de 2022
3.2.0	Importación y exportación de CSV	Ahora puede importar datos de muestra desde un archivo CSV en la herramienta de visualización y también exportar los resultados de una consulta del Generador de operaciones a un archivo CSV.	11 de octubre de 2021
3.1.0	Guardar operaciones	El Generador de operaciones en NoSQL Workbench ahora admite operaciones de guardado para su uso posterior.	12 de julio de 2021

Versión	Cambio	Descripción	Fecha
3.0.0	Configuración de la capacidad e importación/exportación de CloudFormation	NoSQL Workbench para Amazon DynamoDB ahora admite la especificación de un modo de capacidad de lectura/escritura para tablas, y ahora puede importar y exportar modelos de datos en formato CloudFormation.	21 de abril de 2021
2.2.0	Support con PartiQL	NoSQL Workbench para Amazon DynamoDB también admite la creación de instrucciones PartiQL para DynamoDB.	4 de diciembre de 2020
1.1.0	Support con Linux.	NoSQL Workbench para Amazon DynamoDB es compatible con Linux, Ubuntu, Fedora y Debian.	4 de mayo de 2020
1.0.0	NoSQL Workbench para Amazon DynamoDB – GA.	NoSQL Workbench para Amazon DynamoDB se encuentra disponible de manera general.	2 de marzo de 2020

Versión	Cambio	Descripción	Fecha
0.4.1	Compatibilidad con roles de IAM y credenciales de seguridad temporales.	NoSQL Workbench para Amazon DynamoDB ahora admite los roles de AWS Identity and Access Management (IAM) y las credencia les de seguridad temporales.	19 de diciembre de 2019
0.3.1	Compatibilidad con DynamoDB local (versión descargable).	NoSQL Workbench ahora admite conectarse a DynamoDB local (versión descargab le) para diseñar, crear, consultar y administrar tablas de DynamoDB.	8 de noviembre de 2019
0.2.1	Publicación de la vista previa de NoSQL Workbench.	Es la versión inicial de NoSQL Workbench para DynamoDB. Utilice NoSQL Workbench para diseñar, crear, consultar y administr ar tablas DynamoDB.	16 de septiembre de 2019

Copia de seguridad y restauración para DynamoDB

DynamoDB ofrece copias de seguridad bajo demanda y copias de seguridad de recuperación en un momento dado (PITR) para ayudar a proteger los datos de DynamoDB en casos de desastre y ofrece el archivado de datos para la retención de larga duración. Puede crear copias de seguridad de las tablas que ocupen desde unos pocos megabytes a cientos de terabytes de datos, sin impacto alguno en el rendimiento y la disponibilidad de las aplicaciones de producción. Todas las copias de seguridad se cifran y catalogan automáticamente y se pueden detectar fácilmente.

Con las copias de seguridad bajo demanda, puede crear una copia de seguridad instantánea de la tabla que DynamoDB almacena y administra. Se le cobrará en función del tamaño y la duración de las copias de seguridad. Con la copia de seguridad bajo demanda, puede restaurar toda la tabla de DynamoDB al estado exacto en que se encontraba cuando se creó la copia de seguridad.

Existen dos opciones para crear y administrar copias de seguridad bajo demanda de DynamoDB:

- DynamoDB
- AWS Backup

Puede utilizar la capacidad de copia de seguridad bajo demanda de DynamoDB para crear copias de seguridad completas de las tablas con el fin de conservarlas a largo plazo y archivarlas para satisfacer las necesidades de conformidad normativa. Puede realizar una copia de seguridad y restaurar los datos de la tabla en cualquier momento desde la Consola de administración de AWS o con una única llamada a la API.

DynamoDB administra en su totalidad las copias de seguridad de recuperación en un momento dado (PITR) y proporciona hasta 35 días de puntos de recuperación según el grado de detalle por segundo. Para usar la recuperación en un momento dado, que son copias de seguridad continuas, habilite la recuperación en un momento dado (PITR) en la tabla de DynamoDB. Se le cobrará en función del tamaño de la tabla de DynamoDB durante el tiempo que tenga la PITR habilitada en la tabla. La habilitación de la recuperación en un momento dado (PITR) en la tabla de DynamoDB realiza copias de seguridad continuas de los datos. Esto lo ayuda a restaurar la tabla a un punto específico en el tiempo dentro del periodo de recuperación de PITR mediante la creación de una nueva tabla de DynamoDB con el estado exacto de la tabla original en ese momento.

La recuperación a un momento dado ayuda a proteger las tablas de DynamoDB de operaciones accidentales de escritura o eliminación. Al habilitar la recuperación a un momento dado, ya no hay

que preocuparse por crear, mantener o planificar copias de seguridad bajo demanda. Por ejemplo, imaginemos que un script de prueba escribe accidentalmente en una tabla de DynamoDB.

Con la recuperación en un momento dado, puede restaurar la tabla a cualquier momento de los últimos 35 días. Puede establecer el periodo de recuperación en cualquier valor entre 1 y 35 días. Después de habilitar la recuperación en un momento dado, puede restaurar en cualquier momento desde cinco minutos antes del momento actual hasta el periodo de recuperación configurado. DynamoDB mantiene backups acumulativos de la tabla.

Además, las operaciones relacionadas con la recuperación a un momento dado no afectan al desempeño ni a las latencias de la API.

Puede restaurar tablas de DynamoDB a un momento dado mediante la Consola de administración de AWS, la AWS Command Line Interface (AWS CLI) o la API de DynamoDB. El proceso de recuperación a un momento dado siempre restaura la información a una tabla nueva.

Para obtener más información acerca de la disponibilidad y los precios de las regiones de AWS, consulte Precios de Amazon DynamoDB.

> **Note**
>
> - El etiquetado y el control de acceso basado en atributos (ABAC) no se admiten en las copias de seguridad de DynamoDB. Para utilizar ABAC con las copias de seguridad, le recomendamos que utilice AWS Backup.
> - Las etiquetas no se conservan en las tablas restauradas. Debe agregar etiquetas a las tablas restauradas para poder utilizar las condiciones basadas en etiquetas en las políticas.

El siguiente vídeo le ofrece una introducción al concepto de copia de seguridad y restauración y más información sobre la recuperación en el momento.

Copia de seguridad y restauración

Temas

- Copias de seguridad a un momento dado para DynamoDB
- Uso de copia de seguridad y restauración bajo demanda de DynamoDB
- Descripción de la facturación de copias de seguridad en Amazon DynamoDB

- Restauración de una tabla en DynamoDB
- Uso de AWS Backup con DynamoDB

Copias de seguridad a un momento dado para DynamoDB

La recuperación en un momento dado (PITR) de Amazon DynamoDB proporciona copias de seguridad continuas automáticas de los datos de las tablas de DynamoDB. DynamoDB administra en su totalidad las copias de seguridad de recuperación en un momento dado (PITR) y proporciona hasta 35 días de puntos de recuperación según el grado de detalle por segundo. Al habilitar la recuperación a un momento dado, ya no hay que preocuparse por crear, mantener o planificar copias de seguridad bajo demanda. En esta sección se proporciona información general sobre el proceso de trabajo de DynamoDB.

Antes de empezar

Antes de habilitar la recuperación a un momento dado (PITR) en una tabla de Amazon DynamoDB, tenga en cuenta lo siguiente:

- La configuración de `RecoveryPeriodinDays` le permite acortar el periodo de tiempo durante el que se realizan copias de seguridad continuas. De forma predeterminada, los `RecoveryPeriodinDays` son 35. Sin embargo, puede configurarlos para que tengan cualquier valor entre 1 y 35. Acortar los `RecoveryPeriodinDays` no afecta a los precios de PITR porque el precio de basa en el tamaño de la tabla y en los índices secundarios locales.
- Si deshabilita la recuperación a un momento dado y vuelve a habilitarla más tarde en una tabla, estará reiniciando la hora de inicio de recuperación de dicha tabla. Por lo tanto, solo podrá restaurarla inmediatamente con `LatestRestorableDateTime`.
- Puede habilitar la recuperación a un momento dado (PITR) en cada réplica de una misma tabla global. Al restaurar la tabla, el backup se vuelca a una tabla independiente que no es parte de esa tabla global. Si usa la versión 2019.11.21 (actual) de las tablas globales, puede crear una nueva tabla global a partir de la tabla restaurada. Para obtener más información, consulte Funcionamiento de las tablas globales de DynamoDB.
- También puede restaurar los datos de la tabla de DynamoDB en las regiones de AWS de modo que la tabla restaurada se cree en una región distinta de la región en la que reside la tabla de origen. Puede realizar restauraciones entre regiones entre regiones comerciales de AWS, regiones de China de AWS y regiones de AWS GovCloud (EE. UU.). Solo pagará por los datos

que transfiera fuera la de región de origen y por la restauración a una nueva tabla en la región de destino.

- AWS CloudTrail registra todas las acciones de la consola y de la API vinculadas a la recuperación a un momento dado para permitir el mantenimiento de registros, una monitorización continua y auditorías. Para obtener más información, consulte Registrar las operaciones de DynamoDB mediante AWS CloudTrail.

Temas

- Habilitación de la recuperación en un momento dado en DynamoDB

Habilitación de la recuperación en un momento dado en DynamoDB

La recuperación a un momento dado (PITR) en Amazon DynamoDB crea backups automáticos de los datos de tablas de DynamoDB. En esta sección se proporciona información general sobre el proceso de trabajo de DynamoDB.

> **Note**
>
> DynamoDB cobra el PITR en función del tamaño de cada tabla de DynamoDB, incluidos los datos de la tabla y los índices secundarios locales. El cambio del periodo de recuperación (por ejemplo, de 35 días a 1 día) no reduce el precio. El costo sigue siendo el mismo independientemente del periodo de recuperación que elija. El periodo máximo de recuperación configurado no afecta al precio que se le cobra por activar PITR. Para determinar los gastos de copia de seguridad, DynamoDB supervisa de forma continua el tamaño de las tablas que tienen activada la PITR. Se le facturará el uso de la PITR hasta que la desactive para cada tabla.

Temas

- Habilitar la recuperación a un momento dado
- Habilitación de la PITR (consola)
- Habilitación de PITR (AWS CLI)
- Habilitación de PITR (CloudFormation)
- Habilitación de PITR (API)
- Periodo de recuperación

- Edición de PITR
- Eliminación de una tabla con la PITR habilitada

Habilitar la recuperación a un momento dado

Puede habilitar la recuperación a un momento dado mediante la Consola de administración de AWS, la AWS Command Line Interface (AWS CLI) o la API de DynamoDB. Una vez habilitada, la recuperación a un momento dado crea copias de seguridad continuas hasta que la desactive expresamente.

Después de activar la recuperación a un momento dado, puede restaurar a cualquier momento entre `EarliestRestorableDateTime` y `LatestRestorableDateTime`. `LatestRestorableDateTime` es normalmente cinco minutos antes de la hora actual. Para obtener más información, consulte Restauración de una tabla de DynamoDB a un momento específico.

> **Note**
>
> El proceso de recuperación a un momento dado siempre restaura la información a una tabla nueva.

Habilitación de la PITR (consola)

Habilitación de la PITR mediante la consola de DynamoDB

1. Navegue hasta la consola de DynamoDB.
2. Elija Tablas en el menú de navegación de la izquierda y seleccione la tabla de DynamoDB.
3. En la pestaña Copias de seguridad, para la opción Recuperación en un momento dado, elija Editar.
4. Elija Activar la recuperación a un momento dado.
5. Elija un valor entre 1 y 35 para su periodo de recuperación de copia de seguridad. Esto indica el periodo máximo durante el cual se puede recuperar la copia de seguridad continua.

Habilitación de PITR (AWS CLI)

> **Note**
>
> Si recibe errores al ejecutar comandos de la AWS CLI, consulte Solución de errores de la AWS CLI. Asegúrese de que utiliza la versión de la AWS CLI más reciente.

Ejecute el comando update-continuous-backups con la configuración point-in-time-recovery-specification activada:

```
aws dynamodb update-continuous-backups \
--table-name Music \
--point-in-time-recovery-specification
 PointInTimeRecoveryEnabled=true,RecoveryPeriodInDays=35
```

Habilitación de PITR (CloudFormation)

Use el recurso AWS::DynamoDB::Table con la propiedad `PointInTimeRecoverySpecification` activada:

```
Resources:
  iotCatalog:
    Type: AWS::DynamoDB::Table
      Properties:
      ...
      PointInTimeRecoverySpecification:
        PointInTimeRecoveryEnabled: true
        RecoveryPeriodInDays: 35
```

Ejemplo de la sintaxis de la solicitud:

```
{
   "PointInTimeRecoverySpecification": {
      "PointInTimeRecoveryEnabled": boolean,
      "RecoveryPeriodInDays: number
   },
   "TableName": "string"
}
```

Habilitación de PITR (API)

Ejecute la operación de la API UpdateContinuousBackups con el parámetro `PointInTimeRecoverySpecification` activado.

Ejemplo de la sintaxis de la solicitud:

```
{
   "PointInTimeRecoverySpecification": {
      "PointInTimeRecoveryEnabled": boolean,
      "RecoveryPeriodInDays" : number
   },
   "TableName": "string"
}
```

Ejemplo de la sintaxis de la respuesta:

```
{
   "ContinuousBackupsDescription": {
      "ContinuousBackupsStatus": "string",
      "PointInTimeRecoveryDescription": {
         "PointInTimeRecoveryStatus": "string",
         "EarliestRestorableDateTime": number,
         "RecoveryPeriodInDays": number,
         "LatestRestorableDateTime": number
      }
   }
}
```

Python

```
import boto3

dynamodb = boto3.client('dynamodb')

response = dynamodb.update_continuous_backups(
    TableName=<table_name>,
    PointInTimeRecoverySpecification={
        'PointInTimeRecoveryEnabled': True,
        'RecoveryPeriodInDays': 35
    }
)
```

Periodo de recuperación

Puede establecer el periodo de recuperación de copias de seguridad continuas de un valor de entre 1 y 35 días. Estos `RecoveryPeriodInDays` determinan el periodo de tiempo durante el que se mantienen las copias de seguridad continuas. Por ejemplo, si establece este valor en 30 días, solo podrá restaurar la tabla en cualquier momento de los últimos 30 días.

> **Note**
>
> DynamoDB cobra el PITR en función del tamaño de cada tabla de DynamoDB, incluidos los datos de la tabla y los índices secundarios locales. El periodo máximo de recuperación configurado no afecta al precio que se le cobra por activar PITR. Para obtener más información sobre los precios, consulte Precios de DynamoDB.

Edición de PITR

Puede editar la configuración de PITR en la tabla y cambiar el periodo de recuperación. Si cambia el periodo de recuperación y lo aumenta a un valor superior al establecido anteriormente, `EarliestRestorePoint` no cambiará inmediatamente. Como el periodo de recuperación es una ventana móvil, DynamoDB seguirá realizando copias de seguridad automáticas hasta que se alcance el nuevo periodo ampliado. Si cambia el periodo de recuperación y lo disminuye a un valor inferior al establecido anteriormente, `EarliestRestorePoint` disminuirá inmediatamente para coincidir con el periodo de recuperación y cualquier copia de seguridad continua que quede fuera del nuevo valor establecido no será recuperable.

Eliminación de una tabla con la PITR habilitada

Cuando se elimina una tabla que tiene habilitada la recuperación a un momento dado, DynamoDB crea automáticamente una instantánea de copia de seguridad, denominada system backup (copia de seguridad del sistema) y la mantiene durante 35 días (sin costo adicional). Puede utilizar la copia de seguridad del sistema para restaurar la tabla eliminada al estado en el que se encontraba antes de la eliminación. Todas las copias de seguridad del sistema siguen una convención de nomenclatura estándar de *`nombre-tabla`*`$DeletedTableBackup`.

> **Note**
>
> Una vez eliminada una tabla con la recuperación en un momento dado habilitada, puede utilizar la copia de seguridad del sistema para restaurar esa tabla en un único momento

dado. La copia de seguridad del sistema se creará al eliminar la tabla y es una instantánea de la tabla justo antes de que se elimine.

Uso de copia de seguridad y restauración bajo demanda de DynamoDB

Amazon DynamoDB admite características independientes de copia de seguridad y restauración bajo demanda. Esas características están disponibles independientemente de si utiliza AWS Backup.

Puede utilizar la capacidad de backup en diferido de DynamoDB para crear backup completos de las tablas con el fin de conservarlas a largo plazo y archivarlas para satisfacer las necesidades de conformidad normativa. Puede realizar una copia de seguridad y restaurar los datos de la tabla en cualquier momento con un solo clic en la AWS Management Console o con una llamada a la API. Las acciones de backup y restauración se ejecutan sin ningún tipo de impacto en el rendimiento o la disponibilidad de la tabla.

Puede crear copias de seguridad de tabla mediante la consola, la interfaz de línea de comandos de AWS (AWS CLI) o la API de DynamoDB. Para obtener más información, consulte Copia de seguridad de una tabla de DynamoDB.

Para obtener información sobre la restauración de una tabla a partir de un backup, consulte Restauración una tabla de DynamoDB a partir de una copia de seguridad.

Copia de seguridad y restauración de tablas de DynamoDB con DynamoDB: cómo funciona

Puede utilizar la característica de copia de seguridad bajo demanda de DynamoDB para crear copias de seguridad completas de las tablas de Amazon DynamoDB. Esta característica está disponible independientemente de la copia de seguridad de AWS. En esta sección se proporciona información general acerca de qué ocurre durante los procesos de copia de seguridad y restauración de DynamoDB.

Copias de seguridad

Al crear una copia de seguridad bajo demanda con DynamoDB, se cataloga un marcador de tiempo de la solicitud. Para crear el backup de forma asíncrona, todos los cambios realizados hasta el

momento en el que se realiza la solicitud se aplican en la instantánea de tabla completa. Las solicitudes de copia de seguridad de DynamoDB se procesan de forma instantánea y se pueden restaurar en cuestión de minutos.

> **Note**
>
> Cada vez que cree un backup bajo demanda, se crea igualmente un backup con todos los datos de la tabla. No existe ningún límite en cuanto a los backup bajo demanda que se pueden realizar.

Todos los backup de DynamoDB funcionan sin tener que consumir el rendimiento aprovisionado de la tabla.

Los backup de DynamoDB no garantizan la coherencia entre los elementos; no obstante, el sesgo entre actualizaciones en un backup suele ser bastante menos de un segundo.

Mientras se esté realizando un backup, no puede hacer lo siguiente:

- Pausar o cancelar la operación de backup.
- Eliminar la tabla de origen del backup.
- Deshabilitar los backup de una tabla si uno de ellos está en curso.

Si no desea crear scripts de programación y trabajos de limpieza, puede utilizar AWS Backup para crear planes de copia de seguridad con planificaciones y políticas de retención para sus tablas de DynamoDB. AWS Backup ejecuta las copias de seguridad y las elimina cuando caducan. Para obtener más información, consulte la Guía para desarrolladores de AWS Backup.

Además de AWS Backup, puede programar copias de seguridad periódicas o futuras mediante las características de AWS Lambda. Para obtener más información, consulte la publicación de blog A serverless solution to schedule your Amazon DynamoDB On-Demand backup.

Si utiliza la consola, las copias de seguridad creadas utilizando AWS Backup se mostrarán en la pestaña Backups (Copias de seguridad) con el Backup type (Tipo de copia de seguridad) establecido en `AWS`.

> **Note**
>
> No se pueden eliminar backup marcadas con un Tipo Backup de AWS mediante la consola de DynamoDB. Para administrar estas copias de seguridad, utilice la consola de AWS Backup.

Para obtener información sobre cómo realizar un backup, consulte Copia de seguridad de una tabla de DynamoDB.

Restauraciones

Una tabla se restaura sin tener que consumir el rendimiento aprovisionado de la misma. Puede hacer una restauración completa de la tabla desde la backup de DynamoDB o puede configurar los parámetros de la tabla de destino. Al realizar una restauración, puede cambiar la siguiente configuración de tabla:

- Índices secundarios globales (GSI)
- Índices secundarios locales (LSI)
- Modo de facturación:
- Capacidad de lectura y escritura aprovisionada
- Configuración de cifrado

> **Important**
>
> Cuando realiza una restauración de tabla completa, la tabla de destino se establece con las mismas unidades de capacidad de lectura y escritura aprovisionadas que la tabla de origen, ya que se registran en el momento en que se solicita la copia de seguridad. El proceso de restauración también restaura los índices secundarios locales y globales.

También puede restaurar los datos de la tabla de DynamoDB en las regiones de AWS de modo que la tabla restaurada se cree en una región distinta de la región en la que reside el backup. Puede realizar restauraciones entre regiones entre regiones comerciales de AWS, regiones de China de AWS y regiones de AWS GovCloud (EE. UU.). Solo paga por los datos que transfiera fuera de la región de origen y por restaurar a una nueva tabla en la región de destino.

Las restauraciones pueden ser más rápidas y más económicas si elige excluir la creación de algunos o de todos los índices secundarios en la nueva tabla restaurada.

Debe configurar manualmente lo siguiente en la tabla que se restaure:

- Políticas de escalado automático
- Políticas de AWS Identity and Access Management (IAM)
- Alarmas y métricas de Amazon CloudWatch
- Etiquetas
- Ajustes de transmisión
- Configuración del período de vida (TTL)
- Configuración de la protección contra eliminación
- Configuración de la recuperación en un momento dado (PITR)

Solo puede restaurar todos los datos de la tabla en una nueva tabla a partir de un backup. Solo puede escribir en la tabla restaurada después de que se active.

> **Note**
>
> No puede sobrescribir una tabla existente durante una operación de restauración.

Las métricas de servicio muestran que el 95 % de las restauraciones de tablas de los clientes se completan en menos de una hora. Sin embargo, los tiempos de restauración están directamente relacionados con la configuración de las tablas (como el tamaño de las tablas y el número de particiones subyacentes) y otras variables relacionadas. Una práctica recomendada a la hora de planificar la recuperación de desastres es documentar regularmente los tiempos promedio de finalización de la restauración y establecer cómo estos tiempos afectan a su objetivo general de tiempo de recuperación.

Para obtener información sobre cómo realizar una restauración, consulte Restauración una tabla de DynamoDB a partir de una copia de seguridad.

Puede usar las políticas de IAM de control de acceso. Para obtener más información, consulte Uso de IAM con las funcionalidades de copia de seguridad y restauración de DynamoDB.

La consola de backup y restauración y las acciones de API se capturan y registran en AWS CloudTrail para su registro, supervisión y auditoría.

Copia de seguridad de una tabla de DynamoDB

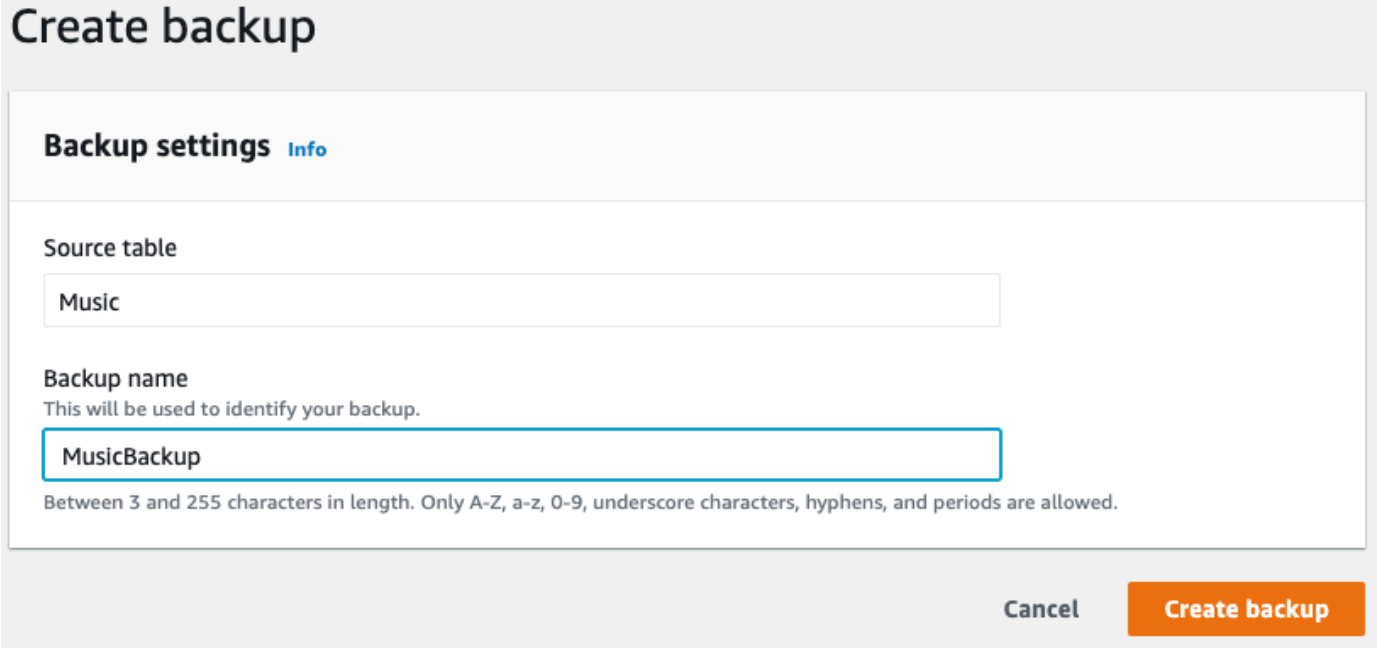
En esta sección se describe cómo utilizar la consola de Amazon DynamoDB o la AWS Command Line Interface para realizar una copia de seguridad de una tabla.

Creación de una copia de seguridad de una tabla (consola)

Siga estos pasos para crear una copia de seguridad con el nombre `MusicBackup` para una tabla `Music` existente con la Consola de administración de AWS.

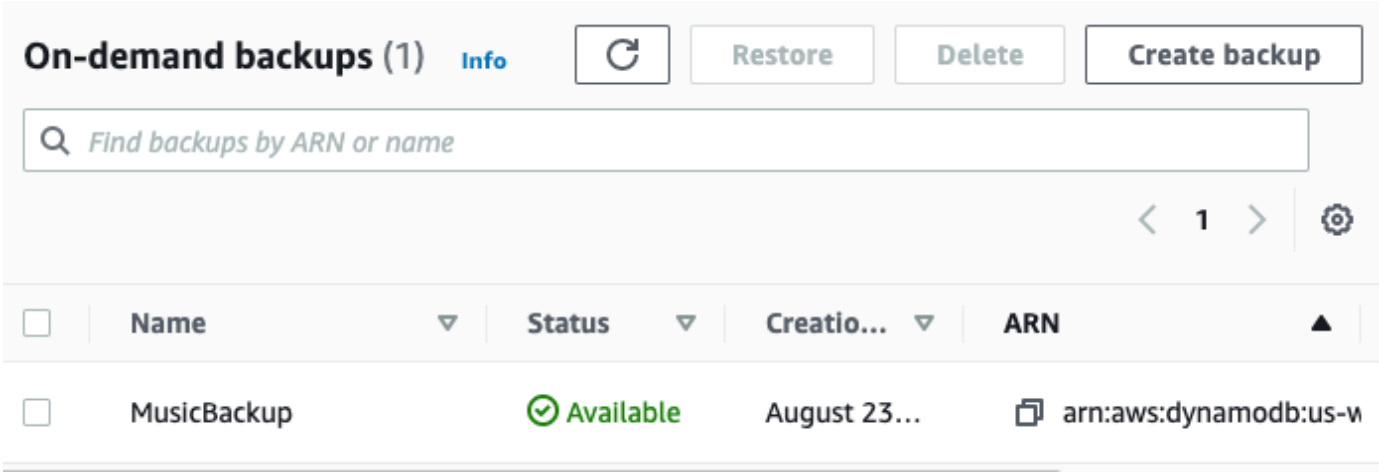
Para crear una copia de seguridad de tabla

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. Puede crear un backup de las siguientes maneras:
   - En la pestaña Backups (Copias de seguridad) de la tabla `Music`, seleccione Create backup (Crear copia de seguridad).
   - En el panel de navegación del lado izquierdo de la consola, elija Backups. A continuación, elija Create backup (Crear backup).
3. Asegúrese de que `Music` es el nombre de la tabla y escriba **`MusicBackup`** para el nombre de la copia de seguridad. A continuación, elija Crear copia de seguridad para crear la copia de seguridad.



> **Note**
>
> Si crea backups usando la sección Backups del panel de navegación, la tabla no se preselecciona automáticamente. Tiene que elegir manualmente el nombre de la tabla de origen para el backup.

Mientras se está creando el backup, su estado se establece en Creating (Creando). Una vez que se haya completado la copia de seguridad, el estado de la misma cambia a Available (Disponible).



Creación de una copia de seguridad de una tabla (AWS CLI)

Siga estos pasos para crear una copia de seguridad de una tabla `Music` existente con AWS CLI.

Para crear una copia de seguridad de tabla

- Cree una copia de seguridad con el nombre `MusicBackup` para la tabla `Music`:

```
aws dynamodb create-backup --table-name Music \
 --backup-name MusicBackup
```

Mientras se está creando la copia de seguridad, su estado se establece en `CREATING`:

```
{
    "BackupDetails": {
        "BackupName": "MusicBackup",
```

```
        "BackupArn": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc",
        "BackupStatus": "CREATING",
        "BackupCreationDateTime": 1489602797.149
    }
}
```

Una vez completada la copia de seguridad, su `BackupStatus` debe cambiar a `AVAILABLE`. Para confirmarlo, use el comando `describe-backup`. Puede obtener el valor de entrada de `backup-arn` desde la salida del paso anterior o usar el comando `list-backups`.

```
aws dynamodb describe-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/backup/01489173575360-
b308cd7d
```

Para realizar un seguimiento de los backups, puede usar el comando `list-backups`. Permite enumerar todas las copias de seguridad que están en estado `CREATING` o `AVAILABLE`:

```
aws dynamodb list-backups
```

El comando `list-backups` y el comando `describe-backup` resultan útiles para comprobar información acerca de la tabla de origen del backup.

Restauración una tabla de DynamoDB a partir de una copia de seguridad

En esta sección se describe cómo restaurar una tabla a partir de un backup mediante la consola de Amazon DynamoDB o mediante la AWS Command Line Interface (AWS CLI).

> **Note**
>
> Si quiere utilizar la AWS CLI, primero tendrá que configurarla. Para obtener más información, consulte Acceso a DynamoDB.

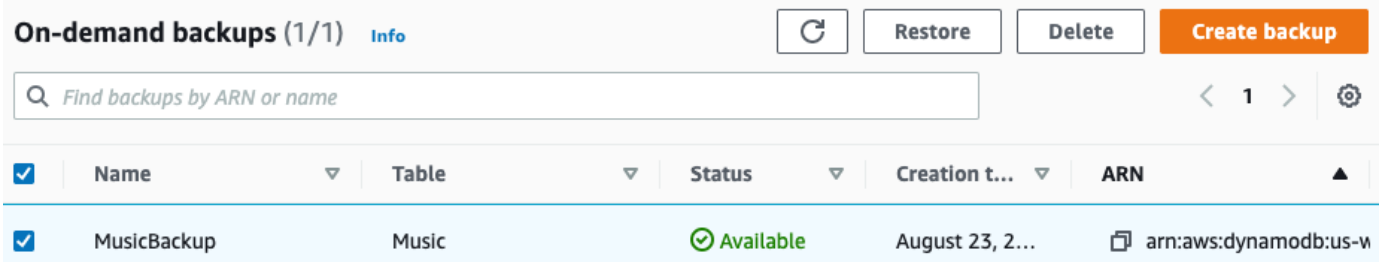
Restauración de una tabla a partir de una copia de seguridad (consola)

En el siguiente procedimiento se muestra cómo restaurar la tabla `Music` mediante el archivo `MusicBackup` que se creó en el tutorial Copia de seguridad de una tabla de DynamoDB.

> **Note**
>
> Este procedimiento asume que la tabla `Music` ya no existe antes de restaurarla utilizando el archivo `MusicBackup`.

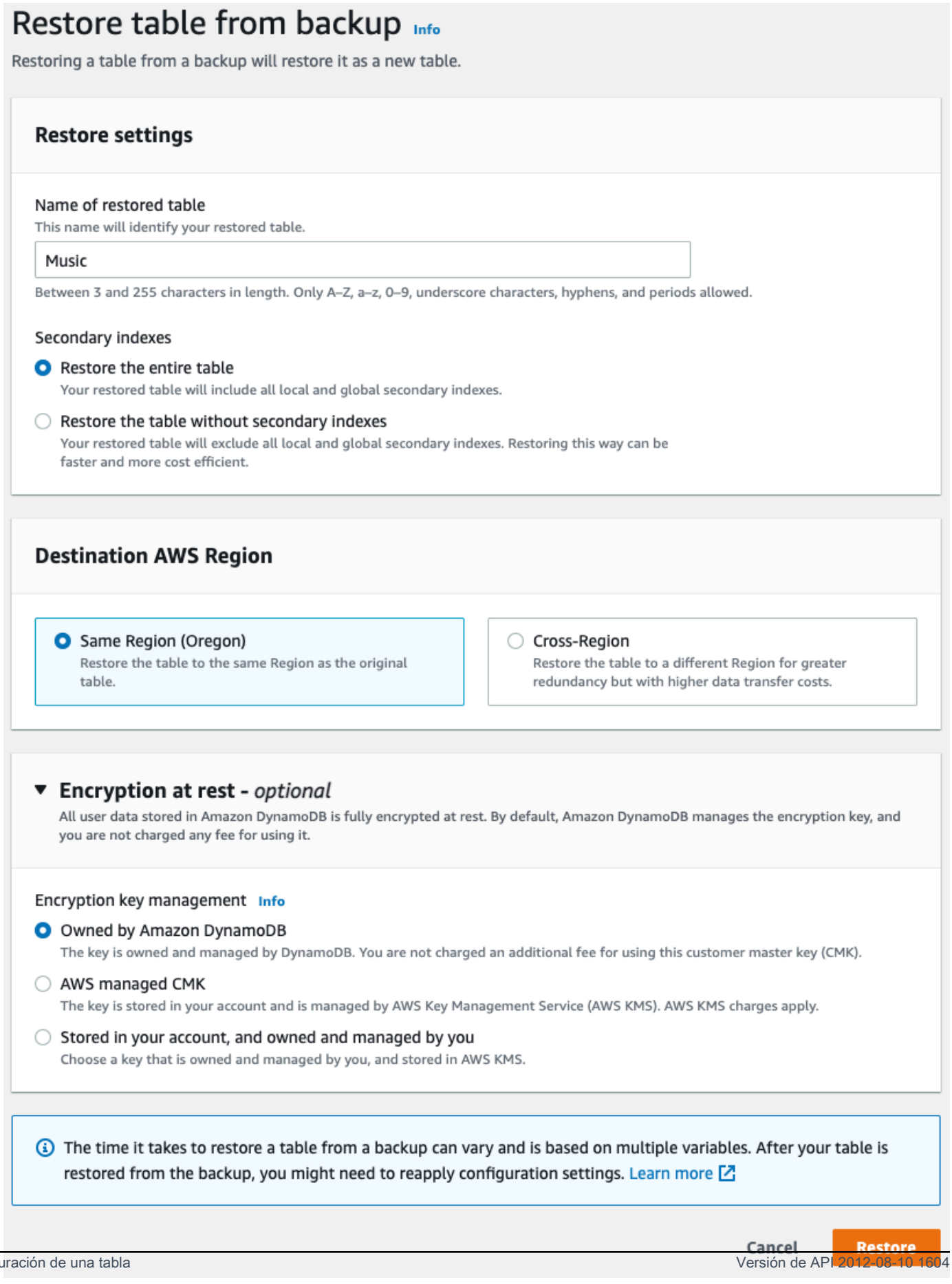
Para restaurar una tabla a partir de una copia de seguridad

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación del lado izquierdo de la consola, elija Backups.
3. En la lista de backups, elija `MusicBackup`.



4. Elija Restore (Restaurar).
5. Especifique **`Music`** como nuevo nombre de tabla. Confirme el nombre del backup y otros detalles del mismo. A continuación, elija Restore table (Restaurar tabla) para iniciar el proceso de restauración.

> **Note**
>
> Puede restaurar la tabla a la misma región de AWS o a otra región distinta a donde reside el backup. También puede excluir la creación de los índices secundarios en la nueva tabla restaurada. Además, puede especificar un modo de cifrado diferente. Las tablas restauradas a partir de copias de seguridad siempre se crean utilizando la clase de tabla DynamoDB Estándar.


Restore table from backup Info
Restoring a table from a backup will restore it as a new table.
Restore settings
Name of restored table
This name will identify your restored table.
Music
Between 3 and 255 characters in length. Only A–Z, a–z, 0–9, underscore characters, hyphens, and periods allowed.
Secondary indexes
Restore the entire table
Your restored table will include all local and global secondary indexes.
Restore the table without secondary indexes
Your restored table will exclude all local and global secondary indexes. Restoring this way can be faster and more cost efficient.
Destination AWS Region
Same Region (Oregon)
Restore the table to the same Region as the original table.
Cross-Region
Restore the table to a different Region for greater redundancy but with higher data transfer costs.
Encryption at rest - optional
All user data stored in Amazon DynamoDB is fully encrypted at rest. By default, Amazon DynamoDB manages the encryption key, and you are not charged any fee for using it.
Encryption key management Info
Owned by Amazon DynamoDB
The key is owned and managed by DynamoDB. You are not charged an additional fee for using this customer master key (CMK).
AWS managed CMK
The key is stored in your account and is managed by AWS Key Management Service (AWS KMS). AWS KMS charges apply.
Stored in your account, and owned and managed by you
Choose a key that is owned and managed by you, and stored in AWS KMS.
The time it takes to restore a table from a backup can vary and is based on multiple variables. After your table is restored from the backup, you might need to reapply configuration settings. Learn more
Cancel
Restore

La tabla que se va a restaurar se muestra con el estado Creating (Creando). Una vez finalizado el proceso de restauración, el estado de la tabla `Music` cambia a Active (Activa).

Restauración de una tabla a partir de una copia de seguridad (AWS CLI)

Siga estos pasos para utilizar la AWS CLI para restaurar la tabla `Music` mediante la opción `MusicBackup` que se creó en el tutorial Copia de seguridad de una tabla de DynamoDB.

Para restaurar una tabla a partir de una copia de seguridad

1. Confirme el backup que quiere restaurar mediante el comando `list-backups`. En este ejemplo se utiliza `MusicBackup`.

   ```
   aws dynamodb list-backups
   ```

   Para obtener detalles adicionales del backup, use el comando `describe-backup`. Puede obtener la entrada `backup-arn` del paso anterior.

   ```
   aws dynamodb describe-backup \
   --backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
   backup/01489173575360-b308cd7d
   ```

2. Restaure la tabla a partir del backup. En este caso, `MusicBackup` restaura la tabla `Music` a la misma región de AWS.

   ```
   aws dynamodb restore-table-from-backup \
   --target-table-name Music \
   --backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
   backup/01489173575360-b308cd7d
   ```

3. Restaure la tabla desde la copia de seguridad con la configuración de tabla personalizada. En este caso, `MusicBackup` restaura la tabla `Music` y especifica un modo de cifrado para la tabla restaurada.

   > **Note**
   >
   > El parámetro `sse-specification-override` toma los mismos valores que el parámetro `sse-specification-override` utilizado en el comando `CreateTable`.

Para obtener más información, consulte Administración de tablas de cifrado en DynamoDB.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581080476474-e177ebe2 \
--sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

Puede restaurar la tabla en una región de AWS distinta a aquella en la que reside el backup.

Note

- El parámetro `sse-specification-override` es obligatorio para restauraciones entre regiones, pero opcional para restauraciones en la misma región que la tabla de origen.
- Al realizar una restauración entre regiones desde la línea de comandos, debe establecer la región de AWS predeterminada en la región de destino deseada. Para obtener más información, consulte Opciones de línea de comandos en la Guía del usuario de AWS Command Line Interface.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581080476474-e177ebe2 \
--sse-specification-override Enabled=true,SSEType=KMS
```

Puede reemplazar el modo de facturación y el rendimiento aprovisionado para la tabla restaurada.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d \
--billing-mode-override PAY_PER_REQUEST
```

Puede excluir la creación de algunos o de todos los índices secundarios en la tabla restaurada.

> **Note**
>
> Las restauraciones pueden ser más rápidas y más económicas si elige excluir la creación de algunos o de todos los índices secundarios en la tabla restaurada.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01581081403719-db9c1f91 \
--global-secondary-index-override '[]' \
--sse-specification-override Enabled=true,SSEType=KMS
```

> **Note**
>
> Los índices secundarios proporcionados deben coincidir con los índices existentes. No puede crear nuevos índices en el momento de la restauración.

Puede utilizar una combinación de distintas anulaciones. Por ejemplo, puede utilizar un único índice secundario global y cambiar el rendimiento aprovisionado al mismo tiempo, como se indica a continuación.

```
aws dynamodb restore-table-from-backup \
--target-table-name Music \
--backup-arn arn:aws:dynamodb:eu-west-1:123456789012:table/Music/
backup/01581082594992-303b6239 \
--billing-mode-override PROVISIONED \
--provisioned-throughput-override ReadCapacityUnits=100,WriteCapacityUnits=100 \
--global-secondary-index-override IndexName=singers-
index,KeySchema=["{AttributeName=SingerName,KeyType=HASH}"],Projection="{ProjectionType=KEYS_
 \
--sse-specification-override Enabled=true,SSEType=KMS
```

Para comprobar la restauración, use el comando `describe-table` para describir la tabla `Music`.

```
aws dynamodb describe-table --table-name Music
```

La tabla que se va a restaurar a partir del backup se muestra con el estado Creating (Creando). Una vez finalizado el proceso de restauración, el estado de la tabla `Music` cambia a Active (Activa).

> **Important**
>
> Mientras la restauración esté en curso, no modifique ni elimine su política de roles de IAM, ya que podría conllevar comportamientos inesperados. Por ejemplo, suponga que eliminó los permisos de escritura de una tabla mientras esta se restauraba. En ese caso, la operación subyacente `RestoreTableFromBackup` no puede escribir ninguno de los datos restaurados en la tabla.
> Después de que se complete la operación de restauración, puede modificar o eliminar su política de roles de IAM.
> Políticas de IAM que implican restricciones de la IP de origen para acceder a la tabla de restauración de destino debe tener el conjunto de clave `aws:ViaAWSService`establecida en `false` para asegurar que las restricciones se apliquen únicamente a las solicitudes presentadas directamente por un principal. De lo contrario, la restauración se cancelará.
> Si la copia de seguridad está cifrada con una Clave administrada de AWS o una clave administrada por el cliente, no desactive ni elimine la clave mientras haya una restauración en curso, pues si lo hace la restauración no se realizará correctamente.
> Una vez finalizada la operación de restauración, puede cambiar la clave de cifrado de la tabla restaurada y deshabilitar o eliminar la clave anterior.

Eliminación de una copia de seguridad de una tabla de DynamoDB

En esta sección se describe cómo utilizar la Consola de administración de AWS o la AWS Command Line Interface (AWS CLI) para eliminar un backup de una tabla de Amazon DynamoDB.

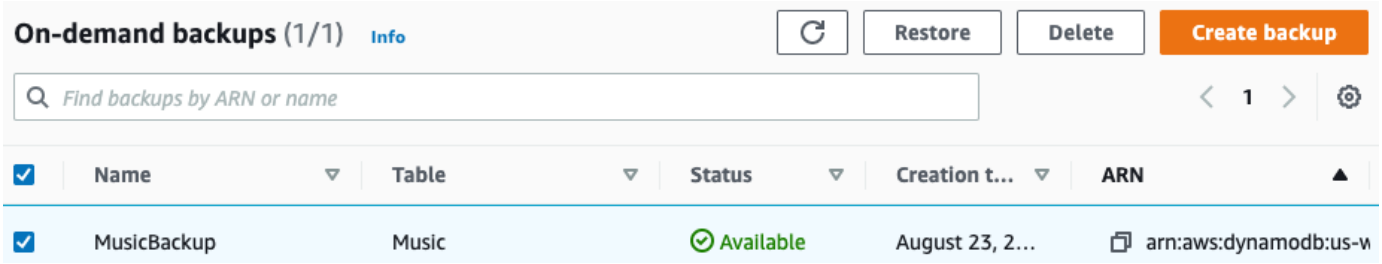
> **Note**
>
> Si quiere utilizar la AWS CLI, primero tendrá que configurarla. Para obtener más información, consulte Uso de AWS CLI.

Eliminación de una copia de seguridad de una tabla (consola)

El siguiente procedimiento muestra cómo usar la consola para eliminar el archivo `MusicBackup` que se ha creado en el tutorial Copia de seguridad de una tabla de DynamoDB.

Para eliminar una copia de seguridad

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación del lado izquierdo de la consola, elija Backups.
3. En la lista de backups, elija `MusicBackup`.



4. Elija Eliminar. Confirme que desea eliminar la copia de seguridad escribiendo **`delete`** y haga clic en Delete (Borrar).

Eliminación de una copia de seguridad de una tabla (AWS CLI)

En el siguiente ejemplo se elimina una copia de seguridad para una tabla existente `Music` mediante la AWS CLI.

```
aws dynamodb delete-backup \
--backup-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489602797149-73d8d5bc
```

Uso de IAM con las funcionalidades de copia de seguridad y restauración de DynamoDB

Puede utilizar AWS Identity and Access Management (IAM) para restringir las acciones de backup y restauración de Amazon DynamoDB para algunos recursos. Las API `CreateBackup` y `RestoreTableFromBackup` operan para cada tabla.

Para obtener más información sobre el uso de políticas de IAM en DynamoDB, consulte Políticas basadas en identidad de DynamoDB.

A continuación, se proporcionan ejemplos de políticas de IAM que puede utilizar para configurar funcionalidades de backup y restauración específicas en DynamoDB.

Ejemplo 1: permitir las acciones CreateBackup y RestoreTableFromBackup

La política de IAM siguiente concede permisos para permitir las acciones `CreateBackup` y `RestoreTableFromBackup` de DynamoDB en todas las tablas:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateBackup",
                "dynamodb:RestoreTableFromBackup",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Important**
>
> Los permisos RestoreTableFromBackup de DynamoDB son necesarios en la copia de seguridad de origen, y los permisos de lectura y escritura de DynamoDB en la tabla de destino son necesarios para la funcionalidad de restauración.
> Los permisos RestoreTableToPointInTime de DynamoDB son necesarios en la tabla de origen, y los permisos de lectura y escritura de DynamoDB en la tabla de destino son necesarios para la funcionalidad de restauración.

Ejemplo 2: permitir la acción CreateBackup y denegar la acción RestoreTableFromBackup

La política de IAM siguiente concede permisos para la acción `CreateBackup` y deniega la acción `RestoreTableFromBackup`:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:RestoreTableFromBackup"],
            "Resource": "*"
        }

    ]
}
```

Ejemplo 3: permitir la acción ListBackups y denegar las acciones CreateBackup y RestoreTableFromBackup

La política de IAM siguiente concede permisos para la acción `ListBackups` y deniega las acciones `CreateBackup` y `RestoreTableFromBackup`:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
```

```
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateBackup",
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "*"
        }

    ]
}
```

Ejemplo 4: permitir la acción ListBackups y denegar la acción DeleteBackup

La política de IAM siguiente concede permisos para la acción `ListBackups` y deniega la acción `DeleteBackup`:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": ["dynamodb:DeleteBackup"],
            "Resource": "*"
        }

    ]
}
```

Ejemplo 5: permitir las acciones RestoreTableFromBackup y DescribeBackup para todos los recursos y denegar la acción DeleteBackup para una copia de seguridad concreta

La siguiente política de IAM concede permisos para las acciones `RestoreTableFromBackup` y `DescribeBackup` y deniega la acción `DeleteBackup` para un recurso de backup concreto:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeBackup",
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:UpdateItem",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:DeleteBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/01489173575360-b308cd7d"
        }
```

```
    ]
}
```

> **Important**
>
> Los permisos RestoreTableFromBackup de DynamoDB son necesarios en la copia de seguridad de origen, y los permisos de lectura y escritura de DynamoDB en la tabla de destino son necesarios para la funcionalidad de restauración.
> Los permisos RestoreTableToPointInTime de DynamoDB son necesarios en la tabla de origen, y los permisos de lectura y escritura de DynamoDB en la tabla de destino son necesarios para la funcionalidad de restauración.

Ejemplo 6: permitir la acción CreateBackup para una tabla concreta

La siguiente política de IAM concede permisos para la acción `CreateBackup` únicamente en la tabla `Movies`:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateBackup"],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/Movies"
            ]
        }
    ]
}
```

Ejemplo 7: Permitir la acción ListBackups

La siguiente política de IAM concede permisos para la acción `ListBackups`:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:ListBackups"],
            "Resource": "*"
        }
    ]
}
```

> **Important**
>
> No puede conceder permisos para la acción `ListBackups` en una tabla concreta.

Ejemplo 8: permitir el acceso a las características de AWS Backup

Necesitará permisos de API para la acción `StartAwsBackupJob` para realizar correctamente una copia de seguridad con características avanzadas y la acción `dynamodb:RestoreTableFromAwsBackup` para restaurar correctamente esa copia de seguridad.

Las siguiente política de IAM concede a AWS Backup los permisos para activar copias de seguridad con características avanzadas y restauraciones. Tenga en cuenta también que si las tablas están cifradas, la política necesitará acceso a la clave de AWS KMS.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeQueryScanBooksTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:StartAwsBackupJob",
                "dynamodb:DescribeTable",
```

```
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Books"
        },
        {
            "Sid": "AllowRestoreFromAwsBackup",
            "Effect": "Allow",
            "Action": [
                "dynamodb:RestoreTableFromAwsBackup"
            ],
            "Resource": "*"
        }
    ]
}
```

Ejemplo 9: Denegar RestoreTableToPointInTime para una tabla de origen específica

La siguiente política de IAM deniega los permisos para la acción `RestoreTableToPointInTime` para una tabla de origen específica:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableToPointInTime"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music"
        }
    ]
}
```

Ejemplo 10: Denegar RestoreTableFromBackup para una tabla de origen específica

La siguiente política de IAM deniega permisos para la acción `RestoreTableToPointInTime` de todas las copias de seguridad correspondiente a una tabla de origen específica:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:RestoreTableFromBackup"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Music/
backup/*"
        }
    ]
}
```

Descripción de la facturación de copias de seguridad en Amazon DynamoDB

Esta guía proporciona detalles sobre cómo funciona la facturación de DynamoDB para las copias de seguridad. Desglosaremos los distintos componentes que contribuyen al coste total y proporcionaremos explicaciones claras y ejemplos prácticos.

DynamoDB ofrece copias de seguridad bajo demanda y copias de seguridad de recuperación en un momento dado (PITR) para ayudar a proteger los datos de DynamoDB en casos de desastre y ofrece el archivado de datos para la retención de larga duración.

Funcionamiento

Las copias de seguridad bajo demanda de DynamoDB se facturan mensualmente. Si realiza una copia de seguridad en un día determinado del mes, verá un cargo único por esa copia de seguridad calculado para los días restantes del mes (por ejemplo, si crea una copia de seguridad el día 27, solo se le cobrará por los pocos días restantes de ese mes y se aplicará como un cargo único el día 27).

Si guarda las copias de seguridad realizadas anteriormente durante los meses siguientes, siempre se aplicará un cargo mensual completo por esa copia de seguridad el día 1. Si la copia de seguridad se elimina antes de que acabe el mes, los cargos se ajustarán en función del uso real.

Por ejemplo, si creó una copia de seguridad el 27 de julio y la mantiene durante el mes de agosto, verá los siguientes cargos por esa copia de seguridad:

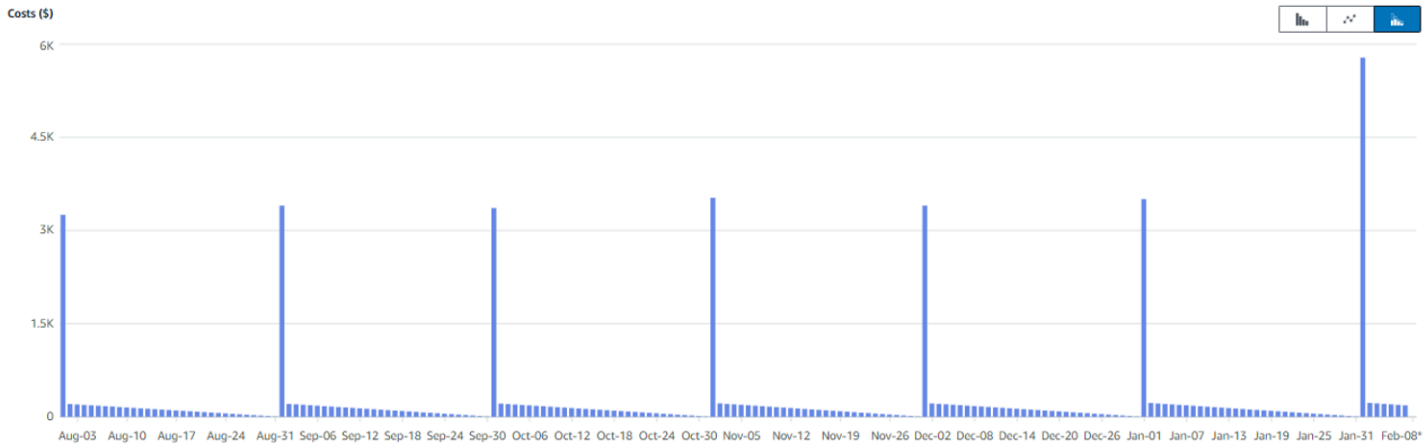
- Un cargo el 27 de julio por los días restantes de julio
- Un cargo el 1 de agosto para todo el mes de agosto
- Un cargo el 1 de cada mes posterior mientras conserve la copia de seguridad

Cuando se mantienen las copias de seguridad de las tablas de DynamoDB, puede observar que el gasto de la métrica de uso `DynamoDB (Region)-TimedBackupStorage-ByteHrs` parece anormalmente alto el primer día del mes. Además, si comprueba esta métrica al principio de un nuevo mes y la compara con los ciclos de facturación anteriores, puede que observe lo que parece ser un gran aumento del uso. Este comportamiento es así por diseño. El primer día de cada mes, se aplicarán cargos de uso por todo el mes a todas las copias de seguridad de DynamoDB existentes. Los gastos de uso de las copias de seguridad de DynamoDB que se eliminen durante el mes se prorratearán para reflejar el uso real. Como resultado, es posible que el cargo (aplicado el día 1) disminuya a lo largo del mes. Esto se debe a que las políticas de retención aplican vencimientos o eliminaciones manuales a las copias de seguridad transferidas de un mes a otro. Esto se analizará en el siguiente escenario.

Del mismo modo, observará picos más pequeños a lo largo del mes a medida que se creen nuevas copias de seguridad, y los cargos se aplicarán el día de la creación durante el resto del mes.

Ejemplo de facturación de copias de seguridad de DynamoDB

A continuación, se muestra un ejemplo de lo que podría ver en Cost Explorer al principio del mes:



Observe cómo el 1 de febrero parece tener un pico mucho mayor en comparación con los meses anteriores. Analicemos por qué ocurre esto.

Desde la página de precios de DynamoDB:

“El tamaño total del almacenamiento de copias de seguridad facturado cada mes es la suma de todas las copias de seguridad de las tablas de DynamoDB. DynamoDB supervisa el tamaño de la copia de seguridad bajo demanda de forma continua durante todo el mes para determinar los cargos de las copias”.

Esto explica por qué la factura muestra un gran aumento del uso todos los día 1 de cada mes. A todas las copias de seguridad existentes que lleguen a un nuevo mes se les aplicará un cargo de mes completo el día 1. Dicho de otro modo, si llega un mes y tiene 300 copias de seguridad de DynamoDB, el primer día del mes se cobrarán los cargos de uso de un mes completo a las 300 copias de seguridad.

Las copias de seguridad nuevas que se realicen a lo largo del mes conllevarán gastos de uso desde el día de su creación hasta el final del mes.

¿Qué sucede si la copia de seguridad se elimina a mediados de mes?

Estas son algunas situaciones a tener en cuenta:

1. Si se elimina una copia de seguridad del mes anterior el día 15 del mes actual, los cargos de uso correspondientes a esa copia de seguridad, que se seguirán aplicando el día 1, se ajustarán al uso real y no al mes completo supuesto que se aplicó anteriormente. En el siguiente ejemplo, se explica esto con más detalle.
2. Al crear una copia de seguridad durante el mes, los cargos de uso del resto del mes se aplican al día en que se creó. Sin embargo, si elimina esta copia de seguridad antes de que finalice el mes,

los cargos de uso se ajustarán para incluir solo los días en que la copia de seguridad estuvo activa y que aún se aplicaron en su fecha de creación.

¿Por qué el uso en el mes actual parece ser mucho mayor el primer día que en los meses anteriores y qué ocurre si elimino las copias de seguridad?

Para responder a esta importante pregunta de dos partes, vamos a crear un escenario de ejemplo con la siguiente información:

- Duración del mes: 30 días
- Frecuencia de copia de seguridad de DynamoDB: 10/día, 300/mes
- Política de retención de copias de seguridad de DynamoDB: 30 días
- Coste por copia de seguridad de DynamoDB: 2 USD/día, 60 USD/mes
- Total del día 1 del mes anterior (`TimedBackupStorage-ByteHrs`, registrado el día 1 del mes actual): 9300 USD
- Total del mes anterior (`TimedBackupStorage-ByteHrs`): 18 600 USD
- Total del día 1 del mes actual (`TimedBackupStorage-ByteHrs`, registrado el día 1): 18 000 USD
- Cambios en el uso de DynamoDB mes a mes: ninguno

Si utilizamos la información anterior, podemos ver que el mes anterior se crearon 300 copias de seguridad con la política de mantenerlas durante 30 días. El primer día de un nuevo mes, todas estas copias de seguridad permanecen intactas, ya que aún no han llegado al final de su período de recuperación. Sin embargo, cada día que pase, los conjuntos de copias de seguridad más antiguos comenzarán a desaparecer, como se muestra a continuación:

Tabla de eliminación de copias de seguridad de DynamoDB

Nuevo mes	Día 1	Día 2	Día 3	Día 4	Día 5
Total de copias de seguridad del mes anterior transferidas	300	290	280	270	260

- El día 1, podemos ver 300 copias de seguridad a 60 USD al mes por copia de seguridad, lo que supone un total de 18 000 de aplicación de `TimedBackupStorage-ByteHrs`. Esto contrasta con el mes anterior, en el que el total del mes fue de 18 600 USD.
- El día 2, 10 de esas copias de seguridad caducarán y se eliminarán. Cuando esto ocurra, el cargo aplicado a esas copias de seguridad se ajustará al uso real y no al uso supuesto. Esto da como resultado que esas 10 copias de seguridad, que antes se cobraban a un precio de 600 USD (10 copias de seguridad × 30 días), se reduzcan a 20 USD (10 copias de seguridad × 1 día).
- Al día siguiente, el siguiente bloque de 10 unidades caducará y se eliminará, por lo que su uso pasará de 30 días a 2 días y su coste se reducirá a 40 USD (10 copias de seguridad × 2 días).

Cada día que pase, empezará a disminuir ese incremento superior al del mes anterior. Si ampliamos esta cifra para cubrir todo el mes, observaremos lo siguiente:

Evolución de los cargos por copia de seguridad de DynamoDB (primer día del mes)

300 copias de seguridad en bloques de 10	Día 1	Día 10	Día 20	Día 30
Bloque 1	600 USD	20 USD	20 USD	20 USD
Bloque 2	600 USD	40 USD	40 USD	40 USD
Bloque 3	600 USD	60 USD	60 USD	60 USD
Bloque 4	600 USD	80 USD	80 USD	80 USD
Bloque 5	600 USD	100 USD	100 USD	100 USD
Bloque 6	600 USD	120 USD	120 USD	120 USD
Bloque 7	600 USD	140 USD	140 USD	140 USD
Bloque 8	600 USD	160 USD	160 USD	160 USD
Bloque 9	600 USD	180 USD	180 USD	180 USD
Bloque 10	600 USD	600 USD	$200	$200
Bloque 11	600 USD	600 USD	220 USD	220 USD

300 copias de seguridad en bloques de 10	Día 1	Día 10	Día 20	Día 30
Bloque 12	600 USD	600 USD	240 USD	240 USD
Bloque 13	600 USD	600 USD	260 USD	260 USD
Bloque 14	600 USD	600 USD	280 USD	280 USD
Bloque 15	600 USD	600 USD	$300	$300
Bloque 16	600 USD	600 USD	320 USD	320 USD
Bloque 17	600 USD	600 USD	340 USD	340 USD
Bloque 18	600 USD	600 USD	360 USD	360 USD
Bloque 19	600 USD	600 USD	380 USD	380 USD
Bloque 20	600 USD	600 USD	600 USD	400 $
Bloque 21	600 USD	600 USD	600 USD	420 USD
Bloque 22	600 USD	600 USD	600 USD	440 USD
Bloque 23	600 USD	600 USD	600 USD	460 USD
Bloque 24	600 USD	600 USD	600 USD	480 USD
Bloque 25	600 USD	600 USD	600 USD	500 USD
Bloque 26	600 USD	600 USD	600 USD	520 USD
Bloque 27	600 USD	600 USD	600 USD	540 USD
Bloque 28	600 USD	600 USD	600 USD	560 USD
Bloque 29	600 USD	600 USD	600 USD	580 USD
Bloque 30	600 USD	600 USD	600 USD	600 USD
Total del día 1 del mes (USD)	18 000 USD	13 500 USD	10 400 USD	9300 USD

A medida que se elimina un bloque nuevo cada día, su uso se ajusta al número de días que existió, en lugar del importe total del mes. Como resultado, al final del mes, los cargos observados el día 1 habrán bajado de los 18 000 USD iniciales a los 9300 USD esperados. Esta cifra, combinada con las copias de seguridad recién creadas a lo largo del mes (que tendrán una tabla de facturación similar a la anterior, pero invertida), generará un gasto mensual acorde a los 18 600 USD del mes anterior.

Restauración de una tabla en DynamoDB

Puede restaurar una tabla de DynamoDB a partir de la copia de seguridad de PITR o de las copias de seguridad bajo demanda mediante la Consola de administración de AWS, la interfaz de línea de comandos de AWS (AWS CLI) o la API de DynamoDB. El proceso de recuperación restaura a una tabla de DynamoDB nueva.

Restauración de una tabla mediante la recuperación a un momento dado

Puede restablecer la tabla a cualquier momento hasta `EarliestRestoreableDateTime`.

> **Important**
>
> Si desactiva la recuperación en un momento dado y la habilita más tarde en una tabla, estará reiniciando la hora de inicio de recuperación de dicha tabla. Por lo tanto, solo podrá restaurarla inmediatamente con `LatestRestorableDateTime`.

Al restaurar mediante la recuperación en un momento dado, DynamoDB restaura los datos de la tabla al estado en el que se encontraban en la fecha y hora seleccionadas (day:hour:minute:second), pero en una tabla nueva. Una tabla se restaura sin tener que consumir el rendimiento aprovisionado de la misma. Puede realizar una restauración completa de la tabla mediante la recuperación a un momento dado o puede configurar los valores de la tabla de destino. Puede cambiar la siguiente configuración de tabla en la tabla restaurada:

- Índices secundarios globales (GSI)
- Índices secundarios locales (LSI)
- Modo de facturación:
- Capacidad de lectura y escritura aprovisionada
- Configuración de cifrado

> **Important**
>
> Cuando realiza una restauración de tabla completa, la tabla de destino se establece con las mismas unidades de capacidad de lectura y escritura aprovisionadas que tenía la tabla de origen cuando se solicitó la copia de seguridad. Por ejemplo, suponga que el rendimiento aprovisionado de una tabla se acaba de reducir a 50 unidades de capacidad de lectura y 50 unidades de capacidad de escritura. A continuación, restaura el estado de la tabla a hace tres semanas, en aquel momento el rendimiento aprovisionado de la tabla era de 100 unidades de capacidad de lectura y 100 unidades de capacidad de escritura. En este caso, DynamoDB restaura los datos de la tabla a ese momento con el rendimiento aprovisionado de ese momento (100 unidades de capacidad de lectura y 100 unidades de capacidad de escritura).

También puede restaurar los datos de la tabla de DynamoDB en las Regiones de AWS de modo que la tabla restaurada se cree en una región distinta de la región en la que reside la tabla de origen. Puede realizar restauraciones entre regiones entre regiones comerciales de AWS, regiones de China de AWS y AWS GovCloud (US). Solo pagará por los datos que transfiera fuera la de región de origen y por la restauración a una nueva tabla en la región de destino.

> **Note**
>
> No se admite la restauración entre regiones si la región de origen o destino es Asia Pacífico (Hong Kong) o Medio Oriente (Baréin).

Las restauraciones pueden ser más rápidas y más rentables si excluye la creación de algunos o de todos los índices en la nueva tabla restaurada. Debe configurar manualmente lo siguiente en la tabla restaurada:

- Políticas de escalado automático
- Políticas de AWS Identity and Access Management
- Métricas y alarmas de eventos de Amazon CloudWatch
- Tags
- Ajustes de transmisión
- Configuración del período de vida (TTL)

- Configuración de la recuperación a un momento dado

El tiempo necesario para restaurar una tabla depende de múltiples factores y no siempre se correlaciona con el tamaño de la tabla.

Restauración de una tabla de DynamoDB a un momento específico

La recuperación a un momento dado (PITR) en Amazon DynamoDB crea backups continuos de los datos de tablas de DynamoDB. Puede restaurar tablas a un momento dado mediante la consola de DynamoDB o la AWS Command Line Interface (AWS CLI). El proceso de recuperación a un momento dado siempre restaura la información a una tabla nueva.

Si quiere utilizar la AWS CLI, primero tendrá que configurarla. Para obtener más información, consulte Acceso a DynamoDB.

Temas

- Restauración de una tabla de DynamoDB a un momento específico (consola)
- Restauración de una tabla a un momento específico (AWS CLI)

Restauración de una tabla de DynamoDB a un momento específico (consola)

En el siguiente ejemplo de código, se muestra cómo utilizar la consola de DynamoDB para restaurar una tabla llamada `Music` a un momento dado.

> **Note**
>
> En este procedimiento se presupone que ha habilitado la recuperación a un momento dado. Para habilitarla para la tabla `Music`, en la pestaña Backups (Copias de seguridad), en la sección Point-in-time recovery (PITR) (Recuperación a un momento dado), elija Edit (Editar) y, a continuación, marque la casilla situada junto a Enable point-in-time-recovery (Habilitar la recuperación a un momento dado).

Para restaurar una tabla a un momento dado

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación del lado izquierdo de la consola, elija Tablas.

3. En la lista de tablas, elija la tabla `Music`.
4. En la pestaña Backups (Copias de seguridad) de la tabla `Music`, en la sección Point-in-time recovery (Recuperación a un momento dado), elija Restore (Restaurar).
5. Para el nombre de tabla nuevo, escriba **`MusicMinutesAgo`**.

   > **Note**
   >
   > Puede restaurar la tabla en la misma región de AWS o en una región distinta a donde reside la tabla de origen. También puede excluir la creación de los índices secundarios en la tabla restaurada. Además, puede especificar un modo de cifrado diferente.

6. Para confirmar el momento de restauración, establezca la fecha y hora de restauración en Earliest (Más temprana). A continuación, elija Restore (Restaurar) para iniciar el proceso de restauración.

   La tabla que se va a restaurar se muestra con el estado Restaurando. Una vez finalizado el proceso de restauración, el estado de la tabla `MusicMinutesAgo` cambia a Active (Activa).

Restauración de una tabla a un momento específico (AWS CLI)

En el siguiente procedimiento, se muestra cómo utilizar la AWS CLI para restaurar una tabla llamada `Music` a un momento dado.

> **Note**
>
> En este procedimiento se presupone que ha habilitado la recuperación a un momento dado. Para habilitarlo en la tabla `Music`, ejecute el siguiente comando.
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name Music \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

Para restaurar una tabla a un momento dado

1. Para confirmar que la recuperación a un momento dado está habilitada en `Music`, utilice el comando `describe-continuous-backups`.

```
aws dynamodb describe-continuous-backups \
    --table-name Music
```

Los backups continuos (se habilitan automáticamente al crear la tabla) y la recuperación a un momento dado están habilitados.

```
{
    "ContinuousBackupsDescription": {
        "PointInTimeRecoveryDescription": {
            "PointInTimeRecoveryStatus": "ENABLED",
            "EarliestRestorableDateTime": 1519257118.0,
            "LatestRestorableDateTime": 1520018653.01
        },
        "ContinuousBackupsStatus": "ENABLED"
    }
}
```

2. Restaure la tabla a un momento dado. En este caso, la tabla `Music` se restaura a `LatestRestorableDateTime` (hace aproximadamente 5 minutos) a la misma región de AWS.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time
```

> **Note**
>
> También puede restaurarla a un momento dado específico. Para ello, ejecute el comando utilizando el argumento `--restore-date-time` y especifique una marca temporal. Puede especificar cualquier momento en el periodo de recuperación configurado, que puede establecerse en cualquier valor entre 1 y 35 días. Por ejemplo, el siguiente comando restaura la tabla al a fecha y hora del valor `EarliestRestorableDateTime`.
>
> ```
> aws dynamodb restore-table-to-point-in-time \
>     --source-table-name Music \
>     --target-table-name MusicEarliestRestorableDateTime \
>     --no-use-latest-restorable-time \
>     --restore-date-time 1519257118.0
> ```

Especificar el argumento `--no-use-latest-restorable-time` es opcional al restaurar datos a un momento dado específico.

3. Restaure la tabla a un momento en el tiempo con la configuración de tabla personalizada. En este caso, la tabla `Music` se restaura a como era en la fecha y hora del valor `LatestRestorableDateTime` (hace más o menos 5 minutos).

   Puede especificar un modo de cifrado distinto para la tabla restaurada, como se indica a continuación.

   > **Note**
   >
   > El parámetro `sse-specification-override` toma los mismos valores que el parámetro `sse-specification-override` utilizado en el comando `CreateTable`. Para obtener más información, consulte Administración de tablas de cifrado en DynamoDB.

   ```
   aws dynamodb restore-table-to-point-in-time \
       --source-table-name Music \
       --target-table-name MusicMinutesAgo \
       --use-latest-restorable-time \
       --sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
   abcd-1234-a123-ab1234a1b234
   ```

   Puede restaurar la tabla en una región de AWS distinta de aquella en la que reside la tabla de origen.

   > **Note**
   >
   > - El parámetro `sse-specification-override` es obligatorio para restauraciones entre regiones, pero opcional para restauraciones en la misma región que la tabla de origen.
   > - El parámetro `source-table-arn` debe proporcionarse para restauraciones entre regiones.
   > - Al realizar una restauración entre regiones desde la línea de comandos, debe establecer la región de AWS predeterminada en la región de destino deseada. Para

obtener más información, consulte Opciones de línea de comandos en la Guía del usuario de AWS Command Line Interface.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-arn arn:aws:dynamodb:us-east-1:123456789012:table/Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --sse-specification-override Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

Puede reemplazar el modo de facturación y el rendimiento aprovisionado para la tabla restaurada.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --billing-mode-override PAY_PER_REQUEST
```

Puede excluir la creación de algunos o de todos los índices secundarios en la tabla restaurada.

> **Note**
>
> Las restauraciones pueden ser más rápidas y más económicas si excluye la creación de algunos o de todos los índices secundarios en la nueva tabla restaurada.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --use-latest-restorable-time \
    --global-secondary-index-override '[]'
```

Puede utilizar una combinación de distintas anulaciones. Por ejemplo, puede utilizar un único índice secundario global y cambiar el rendimiento aprovisionado al mismo tiempo, como se indica a continuación.

```
aws dynamodb restore-table-to-point-in-time \
    --source-table-name Music \
    --target-table-name MusicMinutesAgo \
    --billing-mode-override PROVISIONED \
    --provisioned-throughput-override ReadCapacityUnits=100,WriteCapacityUnits=100
 \
    --global-secondary-index-override IndexName=singers-
index,KeySchema=["{AttributeName=SingerName,KeyType=HASH}"],Projection="{ProjectionType=KEY
 \
    --sse-specification-override Enabled=true,SSEType=KMS \
    --use-latest-restorable-time
```

Para comprobar la restauración, use el comando `describe-table` para describir la tabla `MusicEarliestRestorableDateTime`.

```
aws dynamodb describe-table --table-name MusicEarliestRestorableDateTime
```

La tabla que se va a restaurar se muestra con el estado Creating (Creando) y el valor de restauración en curso true. Una vez finalizado el proceso de restauración, el estado de la tabla `MusicEarliestRestorableDateTime` cambia a Active (Activa).

> **Important**
>
> Mientras haya una restauración en curso, no modifique ni elimine las políticas de AWS Identity and Access Management (IAM) que le conceden a la entidad de (usuario, grupo o rol) permiso para restaurar. De lo contrario, puede ocurrir un comportamiento inesperado. Por ejemplo, suponga que elimina los permisos de escritura de una tabla mientras esta se restauraba. En ese caso, la operación subyacente `RestoreTableToPointInTime` no puede escribir ninguno de los datos restaurados en la tabla. Tenga en cuenta que las políticas de IAM que imponen restricciones a las IP de origen para obtener acceso a la tabla de restauración de destino también pueden causar problemas.
> Puede modificar o eliminar los permisos solo después de que finalice la operación de restauración.

Uso de AWS Backup con DynamoDB

Amazon DynamoDB puede ayudarle a cumplir los requisitos de cumplimiento normativo y de continuidad empresarial gracias a las características mejoradas de las copias de seguridad en AWS Backup. AWS Backup es un servicio de protección de datos completamente administrado que facilita la centralización y automatización de las copias de seguridad en todos los servicios de AWS, en la nube y en las instalaciones. Con este servicio, puede configurar políticas de copia de seguridad y monitorear la actividad de los recursos de AWS en un solo lugar. Para utilizar AWS Backup, debe confirmar la suscripción. Las opciones de suscripción se aplican a la cuenta y región específica de AWS, por lo que es posible que tenga que suscribirse a varias regiones utilizando la misma cuenta. Para obtener más información, consulte la Guía para desarrolladores de AWS Backup.

Amazon DynamoDB se integra de forma nativa con AWS Backup. Puede usar AWS Backup para programar, copiar, etiquetar y realizar el ciclo de vida de sus copias de seguridad bajo demanda de DynamoDB automáticamente. Puede seguir viendo y restaurar estas copias de seguridad desde la consola de DynamoDB. Puede utilizar la consola de DynamoDB, la API y la interfaz de línea de comandos de AWS (AWS CLI) para habilitar las copias de seguridad automáticas de las tablas de DynamoDB.

> **Note**
>
> Las copias de seguridad realizadas a través de DynamoDB permanecerán sin cambios. Podrá seguir creando copias de seguridad mediante el flujo de trabajo actual de DynamoDB.

Las características de copia de seguridad mejoradas disponibles a través de AWS Backup incluyen:

Copias de seguridad programadas: puede configurar copias de seguridad programadas regularmente de las tablas de DynamoDB mediante planes de copia de seguridad.

Copia entre cuentas y regiones: puede copiar automáticamente las copias de seguridad en otros almacén de copias de seguridad en una región o cuenta de AWS diferente, lo que le permite respaldar sus requisitos de protección de datos.

Almacenamiento en frío por niveles: puede configurar las copias de seguridad para implementar reglas de ciclo de vida que permitan eliminar o trasladar las copias de seguridad a un almacenamiento con menos actividad. Esto puede ayudarle a optimizar los costos de las copias de seguridad.

Etiquetas: puede etiquetar automáticamente las copias de seguridad para fines de facturación y asignación de costos.

Cifrado: las copias de seguridad bajo demanda de DynamoDB administradas mediante AWS Backup se almacenan ahora en el almacén de AWS Backup. Esto le permite cifrar y proteger sus copias de seguridad mediante una AWS KMS key independiente de la clave de cifrado de la tabla de DynamoDB.

Auditoría de copias de seguridad: puede utilizar AWS Backup Audit Manager para auditar la conformidad de las políticas de AWS Backup y buscar actividades y recursos de copias de seguridad que aún no cumplen con los controles definidos. También puede usarlo para generar automáticamente un seguimiento de auditoría de informes diarios y bajo demanda para fines de control de copia de seguridad.

Copias de seguridad seguras utilizando el modelo WORM: puede usar el Bloqueo de almacenes de AWS Backup para habilitar una configuración de escritura única y lectura múltiple (WORM) para las copias de seguridad. Con Bloqueo de almacenes de AWS Backup, puede agregar una capa de defensa adicional que protege las copias de seguridad de operaciones de eliminación involuntarias o malintencionadas, cambios en los periodos de recuperación de copias de seguridad y actualizaciones de la configuración del ciclo de vida. Para obtener más información, consulte Bloqueo de almacenes de AWS Backup.

Estas características de copia de seguridad mejoradas están disponibles en todas las regiones de AWS. Para obtener más información sobre estas características, consulte la Guía para desarrolladores de AWS Backup.

Temas

- Copia de seguridad y restauración de las tablas de DynamoDB con AWS Backup: cómo funciona
- Creación de copias de seguridad de las tablas de DynamoDB con AWS Backup
- Copiar una copia de seguridad de una tabla de DynamoDB con AWS Backup
- Restauración de una copia de seguridad de una tabla de DynamoDB desde AWS Backup
- Eliminación de una copia de seguridad de una tabla de DynamoDB con AWS Backup
- Diferencias de nota de uso entre las copias bajo demanda administradas por AWS Backup y DynamoDB

Copia de seguridad y restauración de las tablas de DynamoDB con AWS Backup: cómo funciona

Puede utilizar la característica de backup bajo demanda para crear backup completos de las tablas de Amazon DynamoDB. En esta sección se proporciona información general acerca de qué ocurre durante los procesos de copia de seguridad y restauración.

Copias de seguridad

Al crear una copia de seguridad bajo demanda con AWS Backup, se cataloga un marcador de tiempo de la solicitud. Para crear el backup de forma asíncrona, todos los cambios realizados hasta el momento en el que se realiza la solicitud se aplican en la instantánea de tabla completa.

Cada vez que cree un backup bajo demanda, se crea igualmente un backup con todos los datos de la tabla. No existe ningún límite en cuanto a los backup bajo demanda que se pueden realizar.

> **Note**
>
> A diferencia de las copias de seguridad de DynamoDB, las copias de seguridad realizadas con AWS Backup no son instantáneas.

Mientras se esté realizando un backup, no puede hacer lo siguiente:

- Pausar o cancelar la operación de backup.
- Eliminar la tabla de origen del backup.
- Deshabilitar los backup de una tabla si uno de ellos está en curso.

AWS Backup proporciona programas de copias de seguridad automatizados, administración de retención y administración del ciclo de vida. Esto elimina la necesidad de script personalizados y procesos manuales. AWS Backup ejecuta las copias de seguridad y las elimina cuando vencen. Para obtener más información, consulte la Guía para desarrolladores de AWS Backup.

Si utiliza la consola, las copias de seguridad creadas utilizando AWS Backup se mostrarán en la pestaña Backups (Copias de seguridad) con el Backup type (Tipo de copia de seguridad) establecido en `AWS_BACKUP`.

> **Note**
>
> No se pueden eliminar backup marcadas con un Tipo Backup de `AWS_BACKUP` mediante la consola de DynamoDB. Para administrar estas copias de seguridad, utilice la consola de AWS Backup.

Para obtener información sobre cómo realizar un backup, consulte Copia de seguridad de una tabla de DynamoDB.

Restauraciones

Una tabla se restaura sin tener que consumir el rendimiento aprovisionado de la misma. Puede hacer una restauración completa de la tabla desde la backup de DynamoDB o puede configurar los parámetros de la tabla de destino. Al realizar una restauración, puede cambiar la siguiente configuración de tabla:

- Configuración de cifrado

> **Important**
>
> Cuando realiza una restauración de tabla completa, la tabla de destino se establece con las mismas unidades de capacidad de lectura y escritura aprovisionadas que tenía la tabla de origen cuando se solicitó la copia de seguridad. El proceso de restauración también restaura los índices secundarios locales y globales.

Puede copiar una copia de seguridad de los datos de la tabla de DynamoDB a otra región de AWS y, a continuación, restablecerla en esa nueva región. Puede copiar y restaurar las copias de seguridad entre regiones comerciales de AWS, regiones de AWS de China y regiones GovCloud (EE. UU.) de AWS. Solo pagará por los datos que copie de la región de origen y por los que restaure en una nueva tabla de la región de destino.

AWS Backup restaurará las tablas con todos los índices originales.

Debe configurar manualmente lo siguiente en la tabla que se restaure:

- Políticas de escalado automático

- AWS Identity and Access ManagementPolíticas de (IAM)
- Alarmas y métricas de Amazon CloudWatch
- Etiquetas
- Ajustes de transmisión
- Configuración del período de vida (TTL)
- Configuración de la protección contra eliminación
- Configuración de la recuperación en un momento dado (PITR)

Solo puede restaurar todos los datos de la tabla en una nueva tabla a partir de una copia de seguridad. Solo puede escribir en la tabla restaurada después de que se active.

> **Note**
>
> AWS BackupLas restauraciones de no son destructivas. No puede sobrescribir una tabla existente durante una operación de restauración.

Las métricas de servicio muestran que el 95 % de las restauraciones de tablas de los clientes se completan en menos de una hora. Sin embargo, los tiempos de restauración están directamente relacionados con la configuración de las tablas (como el tamaño de las tablas y el número de particiones subyacentes) y otras variables relacionadas. Una práctica recomendada a la hora de planificar la recuperación de desastres es documentar regularmente los tiempos promedio de finalización de la restauración y establecer cómo estos tiempos afectan a su objetivo general de tiempo de recuperación.

Para obtener información sobre cómo realizar una restauración, consulte Restauración una tabla de DynamoDB a partir de una copia de seguridad.

Puede usar las políticas de IAM de control de acceso. Para obtener más información, consulte Uso de IAM con las funcionalidades de copia de seguridad y restauración de DynamoDB.

La consola de backup y restauración y las acciones de API se capturan y registran en AWS CloudTrail para su registro, supervisión y auditoría.

Creación de copias de seguridad de las tablas de DynamoDB con AWS Backup

En esta sección se describe cómo activar AWS Backup para crear copias de seguridad programadas y bajo demanda desde las tablas de DynamoDB.

Temas

- Activación de las características de AWS Backup
- Copias de seguridad bajo demanda
- Copias de seguridad programadas

Activación de las características de AWS Backup

Debe activar AWS Backup para usarlo con DynamoDB.

Para activar AWS Backup, siga estos pasos:

1. Inicie sesión en AWS Management Console y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación del lado izquierdo de la consola, elija Backups.
3. En la ventana Configuración de copia de seguridad, elija Activar.
4. Aparecerá una pantalla de confirmación. Seleccione Activar características.

Las características de AWS Backup ahora están disponibles para las tablas de DynamoDB.

Si elige desactivar las características de AWS Backup una vez activadas, siga estos pasos:

1. Inicie sesión en AWS Management Console y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación del lado izquierdo de la consola, elija Backups.
3. En la ventana Configuración de copia de seguridad, elija Desactivar.
4. Aparecerá una pantalla de confirmación. Seleccione Desactivar características.

Si no puede activar o desactivar las características de AWS Backup, es posible que el administrador de AWS tenga que realizar estas acciones.

Copias de seguridad bajo demanda

Para crear una copia de seguridad bajo demanda de una tabla de DynamoDB, siga estos pasos:

1. Inicie sesión en AWS Management Console y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación del lado izquierdo de la consola, elija Backups.
3. Elija Create backup.
4. En el menú desplegable que aparece, elija Create an on-demand backup (Creación de una copia de seguridad bajo demanda).
5. Para crear una copia de seguridad administrada por AWS Backup con almacenamiento en caliente y otras características básicas, elija Default Settings (Configuración predeterminada). Para crear una copia de seguridad que se pueda pasar a almacenamiento en frío o crear una copia de seguridad con características de DynamoDB en lugar de AWS Backup, elija Customize settings (Personalizar configuración).

   Si desea crear esta copia de seguridad con características anteriores de DynamoDB, elija Customize settings (Personalizar configuración) y, a continuación, elija Backup with DynamoDB (Copia de seguridad con DynamoDB).
6. Una vez que haya completado la configuración, elijaCreate backup (Crear copia de seguridad).

Copias de seguridad programadas

Para programar una copia de seguridad, siga estos pasos.

1. Inicie sesión en AWS Management Console y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación del lado izquierdo de la consola, elija Backups.
3. En el menú desplegable que aparece, elija Programar copias de seguridad con AWS Backup.
4. Accederá a AWS Backup para crear un plan de copia de seguridad.

Copiar una copia de seguridad de una tabla de DynamoDB con AWS Backup

Puede realizar una copia de una copia de seguridad actual. Puede copiar las copias de seguridad en varias cuentas de AWS o regiones de AWS bajo demanda o de forma automática como parte de

un plan de copias de seguridad programado. También puede automatizar una secuencia de copias entre cuentas y regiones para Amazon DynamoDB Encryption Client.

La replicación entre regiones resulta especialmente útil si hay requisitos de continuidad del negocio o de conformidad que exigen que las copias de seguridad deben almacenarse a una distancia mínima de los datos de producción.

Las copias de seguridad entre cuentas son útiles para copiar de forma segura sus copias de seguridad en una o más cuentas AWS de su organización por motivos operativos o de seguridad. Si la copia de seguridad original se elimina accidentalmente, puede copiar la copia de seguridad de su cuenta de destino a su cuenta de origen y, a continuación, iniciar la restauración. Para poder hacerlo, debe tener dos cuentas que pertenezcan a la misma organización en el servicio Organizations.

Las copias heredan la configuración de la copia de seguridad de origen a menos que especifique lo contrario, con una excepción: si especifica que la nueva copia “Nunca” vence. Con esta configuración, la nueva copia sigue heredando su fecha de vencimiento de origen. Si desea que la nueva copia de seguridad sea permanente, configure las copias de seguridad de origen para que nunca venzan o especifique que la nueva copia venza 100 años después de su creación.

> **Note**
>
> Si va a copiar en otra cuenta, primero debe tener permiso de esa cuenta.

Para copiar una copia de seguridad, haga lo siguiente:

1. Inicie sesión en AWS Management Console y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación del lado izquierdo de la consola, elija Backups.
3. Seleccione la casilla de verificación junto a la copia de seguridad que desea copiar.
   - Si la copia de seguridad que desea copiar aparece atenuada, debe activar las características avanzadas con AWS Backup. A continuación, cree una nueva copia de seguridad. Ahora puede copiar esta nueva copia de seguridad a otras regiones y cuentas, así como cualquier otra copia de seguridad nueva que realice más adelante.
4. Elija Copiar.
5. Si desea copiar la copia de seguridad en otra cuenta o región, seleccione la casilla de verificación situada junto a Copy the recovery point to another destination (Copiar el punto de recuperación

en otro destino). A continuación, seleccione si desea copiar en otra región de su cuenta o en otra cuenta de otra región.

> **Note**
>
> Para restaurar una copia de seguridad en otra región o cuenta, primero debe copiar la copia de seguridad en esa región o cuenta.

6. Seleccione el almacén deseado en el que se copiará el archivo. También puede crear un almacén de copias de seguridad nuevo si lo desea.
7. Elija Copy backup (Copiar copia de seguridad).

Restauración de una copia de seguridad de una tabla de DynamoDB desde AWS Backup

En esta sección se describe cómo restaurar una copia de seguridad de una tabla de DynamoDB desde AWS Backup.

Temas

- Restauración de una tabla de DynamoDB desde AWS Backup
- Restauración de una tabla de DynamoDB en otra región o cuenta

Restauración de una tabla de DynamoDB desde AWS Backup

Para restaurar las tablas de DynamoDB desde AWS Backup, siga estos pasos:

1. Inicie sesión en AWS Management Console y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación del lado izquierdo de la consola, elija Tables (Tablas).
3. Elija la pestaña Backups (Copias de seguridad).
4. Seleccione la casilla de verificación situada junto a la copia de seguridad anterior desde la que desea restaurar.
5. Elija Restore (Restaurar). Accederá a la pantalla Restaurar tabla desde copia de seguridad.
6. Ingrese el nombre de la tabla recién restaurada, el cifrado que tendrá esta nueva tabla, la clave con la que desea cifrar la restauración y otras opciones.

7. Cuando haya terminado, elija Restore (Restaurar).

Restauración de una tabla de DynamoDB en otra región o cuenta

Para restaurar una tabla de DynamoDB en otra región o cuenta, primero tendrá que copiar la copia de seguridad en esa nueva región o cuenta. Para copiar en otra cuenta, esa cuenta primero debe otorgarle permiso. Después de copiar la copia de seguridad de DynamoDB en la nueva región o cuenta, se puede restaurar con el proceso de la sección anterior.

Eliminación de una copia de seguridad de una tabla de DynamoDB con AWS Backup

En esta sección se describe cómo eliminar una copia de seguridad de una tabla de DynamoDB con AWS Backup.

Una copia de seguridad de DynamoDB creada mediante las características de AWS Backup se almacena en un almacén de AWS Backup.

Para eliminar este tipo de copia de seguridad, haga lo siguiente:

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación del lado izquierdo de la consola, elija Backups.
3. En la pantalla que aparece, elija Continuar en AWS Backup.

   Accederá a la Consola de AWS Backup. Para obtener más información sobre cómo eliminar copias de seguridad en la Consola de AWS Backup, consulte Deleting backups.

   Para obtener más información sobre AWS Backup, consulte Copia de seguridad y recuperación mediante AWS Backup en las Recomendaciones de AWS.

Diferencias de nota de uso entre las copias bajo demanda administradas por AWS Backup y DynamoDB

En esta sección se describen las diferencias técnicas entre las copias de seguridad bajo demanda administradas por AWS Backup y DynamoDB.

AWS Backup tiene flujos de trabajo y comportamientos diferentes a los de DynamoDB. Entre ellos se incluyen:

Cifrado: las copias de seguridad creadas con el plan AWS Backup se almacenan en un almacén cifrado con una clave administrada por el servicio de AWS Backup. El almacén tiene políticas de control de acceso para mayor seguridad.

ARN de Backup: los archivos de copia de seguridad creados por AWS Backup ahora dispondrán de un ARN de AWS Backup, que podría afectar al modelo de permisos del usuario. Los nombres de recursos de copia de seguridad (ARN) cambiarán de `arn:aws:dynamodb` a `arn:aws:backup`.

Eliminación de copias de seguridad: las copias de seguridad creadas con AWS Backup solo se puede eliminar del almacén de AWS Backup. No podrá eliminar archivos de AWS Backup de la consola de DynamoDB.

Proceso de copia de seguridad: a diferencia de las copias de seguridad de DynamoDB, las copias de seguridad realizadas con AWS Backup no son instantáneas.

Facturación: las copias de seguridad de tablas de DynamoDB con características de AWS Backup se facturan desde AWS Backup.

Roles de IAM: si administra el acceso a través de roles de IAM, también tendrá que configurar un nuevo rol de IAM con estos nuevos permisos:

```
"dynamodb:StartAwsBackupJob",
"dynamodb:RestoreTableFromAwsBackup"
```

`dynamodb:StartAwsBackupJob` es necesario para realizar una copia de seguridad satisfactoria con características de AWS Backup y se necesita `dynamodb:RestoreTableFromAwsBackup` para restaurar desde una copia de seguridad realizada con características de AWS Backup.

Para ver estos permisos en una política de IAM completa, consulte el ejemplo 8 de Uso de IAM.

Ejemplos de código de DynamoDB con los SDK de AWS

En los siguientes ejemplos de código se muestra cómo utilizar DynamoDB con un kit de desarrollo de software (SDK) de AWS.

Los conceptos básicos son ejemplos de código que muestran cómo realizar las operaciones esenciales dentro de un servicio.

Las acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Mientras las acciones muestran cómo llamar a las distintas funciones de servicio, es posible ver las acciones en contexto en los escenarios relacionados.

Los escenarios son ejemplos de código que muestran cómo llevar a cabo una tarea específica a través de llamadas a varias funciones dentro del servicio o combinado con otros Servicios de AWS.

Las contribuciones de la comunidad de AWS son ejemplos creados y mantenidos por varios equipos de AWS. Para enviar comentarios, utilice el mecanismo previsto en los repositorios vinculados.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Introducción

Introducción a DynamoDB

En los siguientes ejemplos de código, se muestra cómo empezar a utilizar DynamoDB.

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
```

```
using Microsoft.Extensions.DependencyInjection;

namespace DynamoDBActions;

/// <summary>
/// A simple example that demonstrates basic DynamoDB operations.
/// </summary>
public class HelloDynamoDB
{
    /// <summary>
    /// HelloDynamoDB lists the existing DynamoDB tables for the default user.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
            )
            .Build();

        // Now the client is available for injection.
        var dynamoDbClient = host.Services.GetRequiredService<IAmazonDynamoDB>();

        try
        {
            var request = new ListTablesRequest();
            var tableNames = new List<string>();

            var paginatorForTables =
 dynamoDbClient.Paginators.ListTables(request);

            await foreach (var tableName in paginatorForTables.TableNames)
            {
                tableNames.Add(tableName);
            }

            Console.WriteLine("Welcome to the DynamoDB Hello Service example. " +
                              "\nLet's list your DynamoDB tables:");
            tableNames.ForEach(table =>
            {
```

```
                Console.WriteLine($"Table: {table}");
            });
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB service error occurred while
 listing tables. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing tables.
 {ex.Message}");
        }
    }
}
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para .NET.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Código del archivo de CMake CMakeLists.txt.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS dynamodb)

# Set this project's name.
project("hello_dynamodb")
```

```
# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_dynamodb.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Código del archivo de código fuente hello_dynamodb.cpp.

```
#include <aws/core/Aws.h>
#include <aws/dynamodb/DynamoDBClient.h>
#include <aws/dynamodb/model/ListTablesRequest.h>
#include <iostream>
```

```
/*
 *  A "Hello DynamoDB" starter application which initializes an Amazon DynamoDB
 (DynamoDB) client and lists the
 *  DynamoDB tables.
 *
 *  main function
 *
 *  Usage: 'hello_dynamodb'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.

    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::DynamoDB::DynamoDBClient dynamodbClient(clientConfig);
        Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
        listTablesRequest.SetLimit(50);
        do {
            const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
 dynamodbClient.ListTables(
                    listTablesRequest);
            if (!outcome.IsSuccess()) {
                std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
                result = 1;
                break;
            }

            for (const auto &tableName: outcome.GetResult().GetTableNames()) {
                std::cout << tableName << std::endl;
            }

            listTablesRequest.SetExclusiveStartTableName(
                    outcome.GetResult().GetLastEvaluatedTableName());
```

```
        } while (!listTablesRequest.GetExclusiveStartTableName().empty());
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para C++.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListTables {
    public static void main(String[] args) {
```

```
        System.out.println("Listing your Amazon DynamoDB tables:\n");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        listAllTables(ddb);
        ddb.close();
    }

    public static void listAllTables(DynamoDbClient ddb) {
        boolean moreTables = true;
        String lastName = null;

        while (moreTables) {
            try {
                ListTablesResponse response = null;
                if (lastName == null) {
                    ListTablesRequest request =
ListTablesRequest.builder().build();
                    response = ddb.listTables(request);
                } else {
                    ListTablesRequest request = ListTablesRequest.builder()
                            .exclusiveStartTableName(lastName).build();
                    response = ddb.listTables(request);
                }

                List<String> tableNames = response.tableNames();
                if (tableNames.size() > 0) {
                    for (String curName : tableNames) {
                        System.out.format("* %s\n", curName);
                    }
                } else {
                    System.out.println("No tables found!");
                    System.exit(0);
                }

                lastName = response.lastEvaluatedTableName();
                if (lastName == null) {
                    moreTables = false;
                }

            } catch (DynamoDbException e) {
                System.err.println(e.getMessage());
                System.exit(1);
```

```
            }
        }
        System.out.println("\nDone!");
    }
}
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Para obtener más información sobre cómo trabajar con DynamoDB en AWS SDK para JavaScript, consulte Programación de DynamoDB con JavaScript.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import boto3

# Create a DynamoDB client using the default credentials and region
dynamodb = boto3.client("dynamodb")

# Initialize a paginator for the list_tables operation
paginator = dynamodb.get_paginator("list_tables")

# Create a PageIterator from the paginator
page_iterator = paginator.paginate(Limit=10)

# List the tables in the current AWS account
print("Here are the DynamoDB tables in your account:")

# Use pagination to list all tables
table_names = []

for page in page_iterator:
    for table_name in page.get("TableNames", []):
        print(f"- {table_name}")
        table_names.append(table_name)

if not table_names:
    print("You don't have any DynamoDB tables in your account.")
else:
    print(f"\nFound {len(table_names)} tables.")
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
require 'aws-sdk-dynamodb'
require 'logger'

# DynamoDBManager is a class responsible for managing DynamoDB operations
# such as listing all tables in the current AWS account.
class DynamoDBManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Lists and prints all DynamoDB tables in the current AWS account.
  def list_tables
    @logger.info('Here are the DynamoDB tables in your account:')

    paginator = @client.list_tables(limit: 10)
    table_names = []

    paginator.each_page do |page|
      page.table_names.each do |table_name|
        @logger.info("- #{table_name}")
        table_names << table_name
      end
    end

    if table_names.empty?
      @logger.info("You don't have any DynamoDB tables in your account.")
    else
      @logger.info("\nFound #{table_names.length} tables.")
    end
  end
```

```
end

if $PROGRAM_NAME == __FILE__
  dynamodb_client = Aws::DynamoDB::Client.new
  manager = DynamoDBManager.new(dynamodb_client)
  manager.list_tables
end
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para Ruby.

Ejemplos de código

- Ejemplos básicos de DynamoDB con los SDK de AWS
  - Introducción a DynamoDB
  - Obtención de información sobre conceptos básicos de DynamoDB con un SDK de AWS
  - Acciones de DynamoDB con los SDK de AWS
    - Uso de BatchExecuteStatement con un AWS SDK
    - Uso de BatchGetItem con un SDK de AWS o la CLI
    - Uso de BatchWriteItem con un SDK de AWS o la CLI
    - Uso de CreateTable con un SDK de AWS o la CLI
    - Uso de DeleteItem con un SDK de AWS o la CLI
    - Uso de DeleteTable con un SDK de AWS o la CLI
    - Uso de DescribeTable con un SDK de AWS o la CLI
    - Uso de DescribeTimeToLive con un SDK de AWS o la CLI
    - Uso de ExecuteStatement con un SDK de AWS
    - Uso de GetItem con un SDK de AWS o la CLI
    - Uso de ListTables con un SDK de AWS o la CLI
    - Uso de PutItem con un SDK de AWS o la CLI
    - Uso de Query con un SDK de AWS o la CLI
    - Uso de Scan con un SDK de AWS o la CLI
    - Uso de UpdateItem con un SDK de AWS o la CLI
    - Uso de UpdateTable con un SDK de AWS o la CLI

    - Uso de UpdateTimeToLive con un SDK de AWS o la CLI
- Escenarios para DynamoDB con los SDK de AWS
  - Aceleración de lecturas de DynamoDB con DAX con un SDK de AWS
  - Trabajo con escenarios avanzados de índice secundario global de DynamoDB mediante AWS Command Line Interface v2
  - Creación de una aplicación para enviar datos a una tabla de DynamoDB
  - Comparación de varios valores con un solo atributo en DynamoDB con un AWS SDK
  - Actualización condicional de un elemento de DynamoDB con un TTL mediante un AWS SDK
  - Conexión a una instancia local de DynamoDB mediante un SDK de AWS
  - Recuento de operadores de expresión en DynamoDB con un AWS SDK
  - Creación de una API REST de API Gateway para realizar un seguimiento de datos de COVID-19
  - Creación de una aplicación de mensajería con Step Functions
  - Creación de una aplicación de administración de activos fotográficos que permita a los usuarios administrar las fotos mediante etiquetas
  - Creación de una tabla de DynamoDB con un índice secundario global mediante el AWS SDK
  - Creación de una tabla de DynamoDB con configuración de rendimiento en caliente utilizando un AWS SDK
  - Creación de una aplicación web para hacer un seguimiento de los datos de DynamoDB
  - Creación una aplicación de chat de websocket con API Gateway
  - Creación de un elemento de DynamoDB con un TTL mediante un AWS SDK
  - Creación y administración de tablas globales de DynamoDB con coherencia alta de varias regiones mediante un AWS SDK
  - Creación y administración de tablas globales de DynamoDB que muestren MREC con un AWS SDK
  - Eliminación de datos de DynamoDB mediante instrucciones DELETE de PartiQL con un AWS SDK
  - Detección de EPI en imágenes con Amazon Rekognition mediante un AWS SDK
  - Inserción de datos de DynamoDB mediante instrucciones INSERT de PartiQL con un AWS SDK
  - Invocación de una función Lambda desde un navegador
  - Administración de índices secundarios globales de DynamoDB mediante AWS Command Line Interface v2

- Administración de las políticas basadas en recursos de DynamoDB mediante AWS Command Line Interface v2
- Monitoreo del rendimiento de Amazon DynamoDB mediante un SDK de AWS
- Realización de operaciones de consulta avanzadas de DynamoDB mediante un SDK de AWS
- Realización de operaciones de lista en DynamoDB con un AWS SDK
- Realización de operaciones de asignación en DynamoDB con un AWS SDK
- Realización de operaciones de conjunto en DynamoDB con un AWS SDK
- Consultar una tabla de DynamoDB mediante lotes de instrucciones PartiQL y un SDK de AWS
- Consulta de una tabla de DynamoDB con PartiQL y un SDK de AWS
- Consulta de una tabla de DynamoDB mediante un índice secundario global con un SDK de AWS
- Consulta de una tabla de DynamoDB mediante una condición begins_with con un SDK de AWS
- Consulta de una tabla de DynamoDB mediante un intervalo de fechas en la clave de clasificación con un SDK de AWS
- Consulta de una tabla de DynamoDB con una expresión de filtro compleja con un SDK de AWS
- Consulta de una tabla de DynamoDB con una expresión de filtro compleja con un SDK de AWS
- Consulta de una tabla de DynamoDB con una expresión de filtro y límite con un SDK de AWS
- Consulta de una tabla de DynamoDB con atributos anidados mediante un SDK de AWS
- Consulta de una tabla de DynamoDB con paginación mediante un SDK de AWS
- Consulta de una tabla de DynamoDB con lecturas altamente coherentes mediante un SDK de AWS
- Consulta de datos de DynamoDB mediante instrucciones SELECT de PartiQL con un AWS SDK
- Consulta de elementos de TTL en una tabla de DynamoDB con un AWS SDK
- Consulta de tablas de DynamoDB mediante patrones de fecha y hora con un SDK de AWS
- Guarde EXIF y otra información de la imagen con un SDK de AWS
- Configuración del control de acceso basado en atributos para DynamoDB mediante AWS Command Line Interface v2
- Descripción del orden de las expresiones de actualización en DynamoDB con un AWS SDK
- Actualización de una configuración de tabla de DynamoDB con rendimiento en caliente mediante un AWS SDK
- Actualización de un elemento de DynamoDB con un TTL mediante un AWS SDK

- Actualización de datos de DynamoDB mediante instrucciones UPDATE de PartiQL con un AWS SDK
- Uso de API Gateway para invocar una función de Lambda
- Uso de Step Functions para invocar funciones de Lambda
- Utilizar un modelo de documento para DynamoDB mediante un AWS SDK
- Utilizar un modelo de persistencia de objetos de alto nivel para DynamoDB mediante un AWS SDK
- Uso de operaciones de contador atómico en DynamoDB con un AWS SDK
- Uso de operaciones condicionales en DynamoDB con un AWS SDK
- Uso de nombres de atributo de expresión en DynamoDB con un AWS SDK
- Uso de eventos programados para invocar una función de Lambda
- Trabajo con índices secundarios locales de DynamoDB mediante AWS Command Line Interface v2
- Trabajo con DynamoDB Streams y tiempo de vida mediante AWS Command Line Interface v2
- Trabajo con tablas globales de DynamoDB y replicación de varias regiones con coherencia final (MREC) mediante AWS Command Line Interface v2
- Trabajo con el etiquetado de recursos de DynamoDB mediante AWS Command Line Interface v2
- Trabajo con el cifrado de tablas de DynamoDB mediante AWS Command Line Interface v2

- Ejemplos sin servidor para DynamoDB
  - Invocación de una función de Lambda desde un desencadenador de DynamoDB
  - Notificación de los errores de los elementos del lote de las funciones de Lambda con un desencadenador de DynamoDB
- Contribuciones de la comunidad de AWS para DynamoDB
  - Cómo crear y probar una aplicación sin servidor

Ejemplos básicos de DynamoDB con los SDK de AWS

Los siguientes ejemplos de código muestran cómo utilizar conceptos básicos de Amazon DynamoDB con los SDK de AWS.

Ejemplos

- Introducción a DynamoDB
- Obtención de información sobre conceptos básicos de DynamoDB con un SDK de AWS
- Acciones de DynamoDB con los SDK de AWS
  - Uso de BatchExecuteStatement con un AWS SDK
  - Uso de BatchGetItem con un SDK de AWS o la CLI
  - Uso de BatchWriteItem con un SDK de AWS o la CLI
  - Uso de CreateTable con un SDK de AWS o la CLI
  - Uso de DeleteItem con un SDK de AWS o la CLI
  - Uso de DeleteTable con un SDK de AWS o la CLI
  - Uso de DescribeTable con un SDK de AWS o la CLI
  - Uso de DescribeTimeToLive con un SDK de AWS o la CLI
  - Uso de ExecuteStatement con un SDK de AWS
  - Uso de GetItem con un SDK de AWS o la CLI
  - Uso de ListTables con un SDK de AWS o la CLI
  - Uso de PutItem con un SDK de AWS o la CLI
  - Uso de Query con un SDK de AWS o la CLI
  - Uso de Scan con un SDK de AWS o la CLI
  - Uso de UpdateItem con un SDK de AWS o la CLI
  - Uso de UpdateTable con un SDK de AWS o la CLI
  - Uso de UpdateTimeToLive con un SDK de AWS o la CLI

Introducción a DynamoDB

En los siguientes ejemplos de código, se muestra cómo empezar a utilizar DynamoDB.

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Microsoft.Extensions.DependencyInjection;

namespace DynamoDBActions;

/// <summary>
/// A simple example that demonstrates basic DynamoDB operations.
/// </summary>
public class HelloDynamoDB
{
    /// <summary>
    /// HelloDynamoDB lists the existing DynamoDB tables for the default user.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
            )
            .Build();

        // Now the client is available for injection.
        var dynamoDbClient = host.Services.GetRequiredService<IAmazonDynamoDB>();

        try
        {
            var request = new ListTablesRequest();
            var tableNames = new List<string>();

            var paginatorForTables =
 dynamoDbClient.Paginators.ListTables(request);

            await foreach (var tableName in paginatorForTables.TableNames)
            {
                tableNames.Add(tableName);
            }
```

```
            Console.WriteLine("Welcome to the DynamoDB Hello Service example. " +
                              "\nLet's list your DynamoDB tables:");
            tableNames.ForEach(table =>
            {
                Console.WriteLine($"Table: {table}");
            });
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB service error occurred while
 listing tables. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing tables.
 {ex.Message}");
        }
    }
}
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para .NET.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Código del archivo de CMake CMakeLists.txt.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
```

```
set(SERVICE_COMPONENTS dynamodb)

# Set this project's name.
project("hello_dynamodb")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_dynamodb.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Código del archivo de código fuente hello_dynamodb.cpp.

```
#include <aws/core/Aws.h>
#include <aws/dynamodb/DynamoDBClient.h>
#include <aws/dynamodb/model/ListTablesRequest.h>
#include <iostream>

/*
 *  A "Hello DynamoDB" starter application which initializes an Amazon DynamoDB
 (DynamoDB) client and lists the
 *  DynamoDB tables.
 *
 *  main function
 *
 *  Usage: 'hello_dynamodb'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.

    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::DynamoDB::DynamoDBClient dynamodbClient(clientConfig);
        Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
        listTablesRequest.SetLimit(50);
        do {
            const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
 dynamodbClient.ListTables(
                    listTablesRequest);
            if (!outcome.IsSuccess()) {
                std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
                result = 1;
                break;
            }

            for (const auto &tableName: outcome.GetResult().GetTableNames()) {
```

```
                std::cout << tableName << std::endl;
            }

            listTablesRequest.SetExclusiveStartTableName(
                    outcome.GetResult().GetLastEvaluatedTableName());

        } while (!listTablesRequest.GetExclusiveStartTableName().empty());
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para C++.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListTables {
    public static void main(String[] args) {
        System.out.println("Listing your Amazon DynamoDB tables:\n");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        listAllTables(ddb);
        ddb.close();
    }

    public static void listAllTables(DynamoDbClient ddb) {
        boolean moreTables = true;
        String lastName = null;

        while (moreTables) {
            try {
                ListTablesResponse response = null;
                if (lastName == null) {
                    ListTablesRequest request =
 ListTablesRequest.builder().build();
                    response = ddb.listTables(request);
                } else {
                    ListTablesRequest request = ListTablesRequest.builder()
                            .exclusiveStartTableName(lastName).build();
                    response = ddb.listTables(request);
                }

                List<String> tableNames = response.tableNames();
                if (tableNames.size() > 0) {
                    for (String curName : tableNames) {
                        System.out.format("* %s\n", curName);
                    }
                } else {
                    System.out.println("No tables found!");
                    System.exit(0);
                }

                lastName = response.lastEvaluatedTableName();
                if (lastName == null) {
                    moreTables = false;
```

```
                }

            } catch (DynamoDbException e) {
                System.err.println(e.getMessage());
                System.exit(1);
            }
        }
        System.out.println("\nDone!");
    }
}
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Para obtener más información sobre cómo trabajar con DynamoDB en AWS SDK para JavaScript, consulte Programación de DynamoDB con JavaScript.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import boto3

# Create a DynamoDB client using the default credentials and region
dynamodb = boto3.client("dynamodb")

# Initialize a paginator for the list_tables operation
paginator = dynamodb.get_paginator("list_tables")

# Create a PageIterator from the paginator
page_iterator = paginator.paginate(Limit=10)

# List the tables in the current AWS account
print("Here are the DynamoDB tables in your account:")

# Use pagination to list all tables
table_names = []

for page in page_iterator:
    for table_name in page.get("TableNames", []):
        print(f"- {table_name}")
        table_names.append(table_name)

if not table_names:
    print("You don't have any DynamoDB tables in your account.")
else:
    print(f"\nFound {len(table_names)} tables.")
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
require 'aws-sdk-dynamodb'
require 'logger'

# DynamoDBManager is a class responsible for managing DynamoDB operations
# such as listing all tables in the current AWS account.
class DynamoDBManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Lists and prints all DynamoDB tables in the current AWS account.
  def list_tables
    @logger.info('Here are the DynamoDB tables in your account:')

    paginator = @client.list_tables(limit: 10)
    table_names = []

    paginator.each_page do |page|
      page.table_names.each do |table_name|
        @logger.info("- #{table_name}")
        table_names << table_name
      end
    end
```

```
    if table_names.empty?
      @logger.info("You don't have any DynamoDB tables in your account.")
    else
      @logger.info("\nFound #{table_names.length} tables.")
    end
  end
end

if $PROGRAM_NAME == __FILE__
  dynamodb_client = Aws::DynamoDB::Client.new
  manager = DynamoDBManager.new(dynamodb_client)
  manager.list_tables
end
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para Ruby.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Obtención de información sobre conceptos básicos de DynamoDB con un SDK de AWS

En el siguiente ejemplo de código, se muestra cómo:

- Crear una tabla que pueda contener datos de películas.
- Colocar, obtener y actualizar una sola película en la tabla.
- Escribir los datos de películas en la tabla a partir de un archivo JSON de ejemplo.
- Consultar películas que se hayan estrenado en un año determinado.
- Buscar películas que se hayan estrenado en un intervalo de años.
- Eliminar una película de la tabla y, a continuación, eliminar la tabla.

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// This example application performs the following basic Amazon DynamoDB
/// functions:
///     CreateTableAsync
///     PutItemAsync
///     UpdateItemAsync
///     BatchWriteItemAsync
///     GetItemAsync
///     DeleteItemAsync
///     Query
///     Scan
///     DeleteItemAsync.
/// </summary>
public class DynamoDbBasics
{
    public static bool IsInteractive = true;

    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    /// <summary>
    /// The main entry point for the DynamoDB Basics example application.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
```

```
                services.AddAWSService<IAmazonDynamoDB>()
                    .AddTransient<DynamoDbWrapper>())
            .Build();

        // Now the wrapper is available for injection.
        var dynamoDbWrapper =
host.Services.GetRequiredService<DynamoDbWrapper>();

        var tableName = "movie_table";

        var movieFileName = @"movies.json";

        DisplayInstructions();

        // Create a new table and wait for it to be active.
        Console.WriteLine($"Creating the new table: {tableName}");

        var success = await dynamoDbWrapper.CreateMovieTableAsync(tableName);

        Console.WriteLine(success
            ? $"\nTable: {tableName} successfully created."
            : $"\nCould not create {tableName}.");

        WaitForEnter();

        // Add a single new movie to the table.
        var newMovie = new Movie
        {
            Year = 2021,
            Title = "Spider-Man: No Way Home",
        };

        success = await dynamoDbWrapper.PutItemAsync(newMovie, tableName);
        if (success)
        {
            Console.WriteLine($"Added {newMovie.Title} to the table.");
        }
        else
        {
            Console.WriteLine("Could not add movie to table.");
        }

        WaitForEnter();
```

```
        // Update the new movie by adding a plot and rank.
        var newInfo = new MovieInfo
        {
            Plot = "With Spider-Man's identity now revealed, Peter asks" +
                   "Doctor Strange for help. When a spell goes wrong, dangerous"
+
                   "foes from other worlds start to appear, forcing Peter to" +
                   "discover what it truly means to be Spider-Man.",
            Rank = 9,
        };

        success = await dynamoDbWrapper.UpdateItemAsync(newMovie, newInfo,
tableName);
        if (success)
        {
            Console.WriteLine($"Successfully updated the movie:
{newMovie.Title}");
        }
        else
        {
            Console.WriteLine("Could not update the movie.");
        }

        WaitForEnter();

        // Add a batch of movies to the DynamoDB table from a list of
        // movies in a JSON file.
        var itemCount = await dynamoDbWrapper.BatchWriteItemsAsync(movieFileName,
tableName);
        Console.WriteLine($"Added {itemCount} movies to the table.");

        WaitForEnter();

        // Get a movie by key. (partition + sort)
        var lookupMovie = new Movie
        {
            Title = "Jurassic Park",
            Year = 1993,
        };

        Console.WriteLine("Looking for the movie \"Jurassic Park\".");
        var item = await dynamoDbWrapper.GetItemAsync(lookupMovie, tableName);
        if (item?.Count > 0)
        {
```

```
            dynamoDbWrapper.DisplayItem(item);
        }
        else
        {
            Console.WriteLine($"Couldn't find {lookupMovie.Title}");
        }

        WaitForEnter();

        // Delete a movie.
        var movieToDelete = new Movie
        {
            Title = "The Town",
            Year = 2010,
        };

        success = await dynamoDbWrapper.DeleteItemAsync(tableName,
movieToDelete);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {movieToDelete.Title}.");
        }
        else
        {
            Console.WriteLine($"Could not delete {movieToDelete.Title}.");
        }

        WaitForEnter();

        // Use Query to find all the movies released in 2010.
        int findYear = 2010;
        Console.WriteLine($"Movies released in {findYear}");
        var queryCount = await dynamoDbWrapper.QueryMoviesAsync(tableName,
findYear);
        Console.WriteLine($"Found {queryCount} movies released in {findYear}");

        WaitForEnter();

        // Use Scan to get a list of movies from 2001 to 2011.
        int startYear = 2001;
        int endYear = 2011;
        var scanCount = await dynamoDbWrapper.ScanTableAsync(tableName,
startYear, endYear);
```

```
        Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

        WaitForEnter();

        // Delete the table.
        success = await dynamoDbWrapper.DeleteTableAsync(tableName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {tableName}");
        }
        else
        {
            Console.WriteLine($"Could not delete {tableName}");
        }

        Console.WriteLine("The DynamoDB Basics example application is
complete.");

        WaitForEnter();
    }

    /// <summary>
    /// Displays the description of the application on the console.
    /// </summary>
    private static void DisplayInstructions()
    {
        if (!IsInteractive)
        {
            return;
        }

        Console.Clear();
        Console.WriteLine();
        Console.Write(new string(' ', 28));
        Console.WriteLine("DynamoDB Basics Example");
        Console.WriteLine(SepBar);
        Console.WriteLine("This demo application shows the basics of using
DynamoDB with the AWS SDK.");
        Console.WriteLine(SepBar);
        Console.WriteLine("The application does the following:");
        Console.WriteLine("\t1. Creates a table with partition: year and
sort:title.");
```

```
        Console.WriteLine("\t2. Adds a single movie to the table.");
        Console.WriteLine("\t3. Adds movies to the table from moviedata.json.");
        Console.WriteLine("\t4. Updates the rating and plot of the movie that was
 just added.");
        Console.WriteLine("\t5. Gets a movie using its key (partition + sort).");
        Console.WriteLine("\t6. Deletes a movie.");
        Console.WriteLine("\t7. Uses QueryAsync to return all movies released in
 a given year.");
        Console.WriteLine("\t8. Uses ScanAsync to return all movies released
 within a range of years.");
        Console.WriteLine("\t9. Finally, it deletes the table that was just
 created.");
        WaitForEnter();
    }

    /// <summary>
    /// Simple method to wait for the Enter key to be pressed.
    /// </summary>
    private static void WaitForEnter()
    {
        if (IsInteractive)
        {
            Console.WriteLine("\nPress <Enter> to continue.");
            Console.WriteLine(SepBar);
            _ = Console.ReadLine();
        }
    }
}
```

Crea una tabla para contener los datos de las películas.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
```

```
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;
```

```
            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
{tableName}. {ex.Message}");
            throw;
        }
    }
```

Agrega una sola película a la tabla.

```
    /// <summary>
    /// Adds a new item to the table.
```

```
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
 {ex.Message}");
            throw;
        }
    }
```

Actualiza un solo elemento de una tabla.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };
```

```
            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
 {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
 {ex.Message}");
            throw;
        }
    }
```

Recupera un solo elemento de la tabla de películas.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</
param>
```

```
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
            throw;
        }
    }
```

Escribe un lote de elementos en la tabla de películas.

```
/// <summary>
/// Loads the contents of a JSON file into a list of movies to be
/// added to the DynamoDB table.
/// </summary>
/// <param name="movieFileName">The name of the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public List<Movie> ImportMovies(string movieFileName)
{
    var moviesList = new List<Movie>();
    if (!File.Exists(movieFileName))
    {
        return moviesList;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonSerializer.Deserialize<List<Movie>>(
        json,
        new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        });

    // Now return the first 250 entries.
    if (allMovies != null && allMovies.Any())
    {
        moviesList = allMovies.GetRange(0, 250);
    }
    return moviesList;
}

/// <summary>
/// Writes 250 items to the movie table.
/// </summary>
/// <param name="movieFileName">A string containing the full path to
/// the JSON file containing movie data.</param>
/// <param name="tableName">The name of the table to write items to.</param>
/// <returns>A long integer value representing the number of movies
/// imported from the JSON file.</returns>
public async Task<long> BatchWriteItemsAsync(
    string movieFileName, string tableName)
```

```
   {
       try
       {
           var movies = ImportMovies(movieFileName);
           if (!movies.Any())
           {
               Console.WriteLine("Couldn't find the JSON file with movie
data.");
               return 0;
           }

           var context = new DynamoDBContextBuilder()
               // Optional call to provide a specific instance of
IAmazonDynamoDB
               .WithDynamoDBClient(() => _amazonDynamoDB)
               .Build();

           var movieBatch = context.CreateBatchWrite<Movie>(
               new BatchWriteConfig()
               {
                   OverrideTableName = tableName
               });
           movieBatch.AddPutItems(movies);

           Console.WriteLine("Adding imported movies to the table.");
           await movieBatch.ExecuteAsync();

           return movies.Count;
       }
       catch (ResourceNotFoundException ex)
       {
           Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
           throw;
       }
       catch (AmazonDynamoDBException ex)
       {
           Console.WriteLine($"An Amazon DynamoDB error occurred during batch
write operation. {ex.Message}");
           throw;
       }
       catch (Exception ex)
       {
```

```
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

Elimina un solo elemento de la tabla.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N =
movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
```

```
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

Consulta en la tabla las películas estrenadas en un año determinado.

```
    /// <summary>
    /// Queries the table for movies released in a particular year and
    /// then displays the information for the movies returned.
    /// </summary>
    /// <param name="tableName">The name of the table to query.</param>
    /// <param name="year">The release year for which we want to
    /// view movies.</param>
    /// <returns>The number of movies that match the query.</returns>
    public async Task<int> QueryMoviesAsync(string tableName, int year)
    {
        try
        {
            var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
                .AddHashKey("year", DynamoDBEntryType.Numeric)
                .AddRangeKey("title", DynamoDBEntryType.String)
                .Build();

            var filter = new QueryFilter("year", QueryOperator.Equal, year);

            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
```

```
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine($"An error occurred while querying movies.
 {ex.Message}");
            throw;
        }
    }
```

Busca en la tabla películas que se hayan estrenado en un intervalo de años.

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</
returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string,
 AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0],
 info.directors, info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the
 LastEvaluatedKey.
```

```
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
            throw;
        }
    }
```

Elimina la tabla de películas.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
```

```
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
 successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
 deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
 table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
 {tableName}. {ex.Message}");
            return false;
        }
    }
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK para .NET.
  - BatchWriteItem

- CreateTable
- DeleteItem
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Query
- Examen
- UpdateItem

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

El escenario de introducción a DynamoDB.

```
###############################################################################
# function dynamodb_getting_started_movies
#
# Scenario to create an Amazon DynamoDB table and perform a series of operations
 on the table.
#
# Returns:
#       0 - If successful.
#       1 - If an error occurred.
###############################################################################
function dynamodb_getting_started_movies() {

  source ./dynamodb_operations.sh

  key_schema_json_file="dynamodb_key_schema.json"
  attribute_definitions_json_file="dynamodb_attr_def.json"
```

```
  item_json_file="movie_item.json"
  key_json_file="movie_key.json"
  batch_json_file="batch.json"
  attribute_names_json_file="attribute_names.json"
  attributes_values_json_file="attribute_values.json"

  echo_repeat "*" 88
  echo
  echo "Welcome to the Amazon DynamoDB getting started demo."
  echo
  echo_repeat "*" 88
  echo

  local table_name
  echo -n "Enter a name for a new DynamoDB table: "
  get_input
  table_name=$get_input_result

  echo '[
  {"AttributeName": "year", "KeyType": "HASH"},
   {"AttributeName": "title", "KeyType": "RANGE"}
  ]' >"$key_schema_json_file"

  echo '[
  {"AttributeName": "year", "AttributeType": "N"},
   {"AttributeName": "title", "AttributeType": "S"}
  ]' >"$attribute_definitions_json_file"

  if dynamodb_create_table -n "$table_name" -a "$attribute_definitions_json_file"
\
    -k "$key_schema_json_file" 1>/dev/null; then
    echo "Created a DynamoDB table named $table_name"
  else
    errecho "The table failed to create. This demo will exit."
    clean_up
    return 1
  fi

  echo "Waiting for the table to become active...."

  if dynamodb_wait_table_active -n "$table_name"; then
    echo "The table is now active."
  else
    errecho "The table failed to become active. This demo will exit."
```

```
    cleanup "$table_name"
    return 1
  fi

  echo
  echo_repeat "*" 88
  echo

  echo -n "Enter the title of a movie you want to add to the table: "
  get_input
  local added_title
  added_title=$get_input_result

  local added_year
  get_int_input "What year was it released? "
  added_year=$get_input_result

  local rating
  get_float_input "On a scale of 1 - 10, how do you rate it? " "1" "10"
  rating=$get_input_result

  local plot
  echo -n "Summarize the plot for me: "
  get_input
  plot=$get_input_result

  echo '{
    "year": {"N" :"'"$added_year"'"},
    "title": {"S" :  "'"$added_title"'"},
    "info": {"M" : {"plot": {"S" : "'"$plot"'"}, "rating":
{"N" :"'"$rating"'"} } }
   }' >"$item_json_file"

  if dynamodb_put_item -n "$table_name" -i "$item_json_file"; then
    echo "The movie '$added_title' was successfully added to the table
'$table_name'."
  else
    errecho "Put item failed. This demo will exit."
    clean_up "$table_name"
    return 1
  fi

  echo
  echo_repeat "*" 88
```

```
  echo

  echo "Let's update your movie '$added_title'."
  get_float_input "You rated it $rating, what new rating would you give it? " "1"
"10"
  rating=$get_input_result

  echo -n "You summarized the plot as '$plot'."
  echo "What would you say now? "
  get_input
  plot=$get_input_result

  echo '{
    "year": {"N" :"'"$added_year"'"},
    "title": {"S" :  "'"$added_title"'"}
    }' >"$key_json_file"

  echo '{
    ":r": {"N" :"'"$rating"'"},
    ":p": {"S" : "'"$plot"'"}
   }' >"$item_json_file"

  local update_expression="SET info.rating = :r, info.plot = :p"

  if dynamodb_update_item -n "$table_name" -k "$key_json_file" -e
"$update_expression" -v "$item_json_file"; then
    echo "Updated '$added_title' with new attributes."
  else
    errecho "Update item failed. This demo will exit."
    clean_up "$table_name"
    return 1
  fi

  echo
  echo_repeat "*" 88
  echo

  echo "We will now use batch write to upload 150 movie entries into the table."

  local batch_json
  for batch_json in movie_files/movies_*.json; do
    echo "{ \"$table_name\" : $(<"$batch_json") }" >"$batch_json_file"
    if dynamodb_batch_write_item -i "$batch_json_file" 1>/dev/null; then
      echo "Entries in $batch_json added to table."
```

```
    else
      errecho "Batch write failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi
  done

  local title="The Lord of the Rings: The Fellowship of the Ring"
  local year="2001"

  if get_yes_no_input "Let's move on...do you want to get info about '$title'?
(y/n) "; then
    echo '{
  "year": {"N" :"'"$year"'"},
  "title": {"S" :  "'"$title"'"}
  }' >"$key_json_file"
    local info
    info=$(dynamodb_get_item -n "$table_name" -k "$key_json_file")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
      errecho "Get item failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi

    echo "Here is what I found:"
    echo "$info"
  fi

  local ask_for_year=true
  while [[ "$ask_for_year" == true ]]; do
    echo "Let's get a list of movies released in a given year."
    get_int_input "Enter a year between 1972 and 2018: " "1972" "2018"
    year=$get_input_result
    echo '{
    "#n": "year"
    }' >"$attribute_names_json_file"

    echo '{
    ":v": {"N" :"'"$year"'"}
    }' >"$attributes_values_json_file"
```

```
    response=$(dynamodb_query -n "$table_name" -k "#n=:v" -a
"$attribute_names_json_file" -v "$attributes_values_json_file")

    # shellcheck disable=SC2181
    if [[ ${?} -ne 0 ]]; then
      errecho "Query table failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi

    echo "Here is what I found:"
    echo "$response"

    if ! get_yes_no_input "Try another year? (y/n) "; then
      ask_for_year=false
    fi
  done

  echo "Now let's scan for movies released in a range of years. Enter a year: "
  get_int_input "Enter a year between 1972 and 2018: " "1972" "2018"
  local start=$get_input_result

  get_int_input "Enter another year: " "1972" "2018"
  local end=$get_input_result

  echo '{
    "#n": "year"
    }' >"$attribute_names_json_file"

  echo '{
    ":v1": {"N" : "'"$start"'"},
    ":v2": {"N" : "'"$end"'"}
    }' >"$attributes_values_json_file"

  response=$(dynamodb_scan -n "$table_name" -f "#n BETWEEN :v1 AND :v2" -a
"$attribute_names_json_file" -v "$attributes_values_json_file")

  # shellcheck disable=SC2181
  if [[ ${?} -ne 0 ]]; then
    errecho "Scan table failed. This demo will exit."
    clean_up "$table_name"
    return 1
  fi
```

```
  echo "Here is what I found:"
  echo "$response"

  echo
  echo_repeat "*" 88
  echo

  echo "Let's remove your movie '$added_title' from the table."

  if get_yes_no_input "Do you want to remove '$added_title'? (y/n) "; then
    echo '{
  "year": {"N" :"'"$added_year"'"},
  "title": {"S" :  "'"$added_title"'"}
  }' >"$key_json_file"

    if ! dynamodb_delete_item -n "$table_name" -k "$key_json_file"; then
      errecho "Delete item failed. This demo will exit."
      clean_up "$table_name"
      return 1
    fi
  fi

  if get_yes_no_input "Do you want to delete the table '$table_name'? (y/n) ";
 then
    if ! clean_up "$table_name"; then
      return 1
    fi
  else
    if ! clean_up; then
      return 1
    fi
  fi

  return 0
}
```

Las funciones de DynamoDB utilizadas en este escenario.

```
###############################################################################
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
```

```
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table with on-demand billing."
    echo " -n table_name  -- The name of the table to create."
    echo " -a attribute_definitions -- JSON file path of a list of attributes and
 their types."
    echo " -k key_schema -- JSON file path of a list of attributes and their key
 types."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:a:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      a) attribute_definitions="${OPTARG}" ;;
      k) key_schema="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
```

```
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$attribute_definitions" ]]; then
    errecho "ERROR: You must provide an attribute definitions json file path the
-a parameter."
    usage
    return 1
  fi

  if [[ -z "$key_schema" ]]; then
    errecho "ERROR: You must provide a key schema json file path the -k
parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    attribute_definitions:   $attribute_definitions"
  iecho "    key_schema:   $key_schema"
  iecho ""

  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --billing-mode PAY_PER_REQUEST \
    --key-schema file://"$key_schema" )

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi
```

```
  return 0
}

############################################################################
# function dynamodb_describe_table
#
# This function returns the status of a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#
#  Response:
#       - TableStatus:
#     And:
#       0 - Table is active.
#       1 - If it fails.
############################################################################
function dynamodb_describe_table {
  local table_name
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_describe_table"
    echo "Describe the status of a DynamoDB table."
    echo "  -n table_name  -- The name of the table."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
```

```
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  local table_status
    table_status=$(
      aws dynamodb describe-table \
        --table-name "$table_name" \
        --output text \
        --query 'Table.TableStatus'
    )

   local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
      aws_cli_error_log "$error_code"
      errecho "ERROR: AWS reports describe-table operation failed.$table_status"
      return 1
    fi

  echo "$table_status"

  return 0
}

###############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
```

```
###############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }

  while getopts "n:i:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi
```

```
  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""
  iecho ""

  response=$(aws dynamodb put-item \
    --table-name "$table_name" \
    --item file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports put-item operation failed.$response"
    return 1
  fi

  return 0

}

###############################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.
```

```
 ######################################
 # Function usage explanation
 ######################################
 function usage() {
   echo "function dynamodb_update_item"
   echo "Update an item in a DynamoDB table."
   echo " -n table_name  -- The name of the table."
   echo " -k keys  -- Path to json file containing the keys that identify the
item to update."
   echo " -e update expression  -- An expression that defines one or more
attributes to be updated."
   echo " -v values  -- Path to json file containing the update values."
   echo ""
 }

 while getopts "n:k:e:v:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) keys="${OPTARG}" ;;
     e) update_expression="${OPTARG}" ;;
     v) values="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$keys" ]]; then
   errecho "ERROR: You must provide a keys json file path the -k parameter."
   usage
```

```
    return 1
  fi
  if [[ -z "$update_expression" ]]; then
    errecho "ERROR: You must provide an update expression with the -e parameter."
    usage
    return 1
  fi

  if [[ -z "$values" ]]; then
    errecho "ERROR: You must provide a values json file path the -v parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho "    update_expression:   $update_expression"
  iecho "    values:   $values"

  response=$(aws dynamodb update-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --update-expression "$update_expression" \
    --expression-attribute-values file://"$values")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
  fi

  return 0

}

########################################################################################
# function dynamodb_batch_write_item
#
# This function writes a batch of items into a DynamoDB table.
#
# Parameters:
```

```
#       -i item  -- Path to json file containing the items to write.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_batch_write_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_batch_write_item"
    echo "Write a batch of items into a DynamoDB table."
    echo " -i item  -- Path to json file containing the items to write."
    echo ""
  }
  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
```

```
  iecho ""

  response=$(aws dynamodb batch-write-item \
    --request-items file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  return 0
}

###############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_get_item"
    echo "Get an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
```

```
    echo " -k keys  -- Path to json file containing the keys that identify the
item to get."
    echo " [-q query]  -- Optional JMESPath query expression."
    echo ""
 }
 query=""
 while getopts "n:k:q:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      q) query="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
 fi

 if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
 fi

 if [[ -n "$query" ]]; then
    response=$(aws dynamodb get-item \
      --table-name "$table_name" \
      --key file://"$keys" \
      --output text \
      --query "$query")
 else
    response=$(
```

```
      aws dynamodb get-item \
        --table-name "$table_name" \
        --key file://"$keys" \
        --output text
    )
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
  fi

  if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}

###############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_query() {
```

```
 local table_name key_condition_expression attribute_names attribute_values
projection_expression response
 local option OPTARG # Required to use getopts command in a function.

 # ######################################
 # Function usage explanation
 ######################################
 function usage() {
   echo "function dynamodb_query"
   echo "Query a DynamoDB table."
   echo " -n table_name  -- The name of the table."
   echo " -k key_condition_expression -- The key condition expression."
   echo " -a attribute_names -- Path to JSON file containing the attribute
names."
   echo " -v attribute_values -- Path to JSON file containing the attribute
values."
   echo " [-p projection_expression]  -- Optional projection expression."
   echo ""
 }

 while getopts "n:k:a:v:p:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) key_condition_expression="${OPTARG}" ;;
     a) attribute_names="${OPTARG}" ;;
     v) attribute_values="${OPTARG}" ;;
     p) projection_expression="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
```

```
 fi

 if [[ -z "$key_condition_expression" ]]; then
   errecho "ERROR: You must provide a key condition expression with the -k
parameter."
   usage
   return 1
 fi

 if [[ -z "$attribute_names" ]]; then
   errecho "ERROR: You must provide a attribute names with the -a parameter."
   usage
   return 1
 fi

 if [[ -z "$attribute_values" ]]; then
   errecho "ERROR: You must provide a attribute values with the -v parameter."
   usage
   return 1
 fi

 if [[ -z "$projection_expression" ]]; then
   response=$(aws dynamodb query \
     --table-name "$table_name" \
     --key-condition-expression "$key_condition_expression" \
     --expression-attribute-names file://"$attribute_names" \
     --expression-attribute-values file://"$attribute_values")
 else
   response=$(aws dynamodb query \
     --table-name "$table_name" \
     --key-condition-expression "$key_condition_expression" \
     --expression-attribute-names file://"$attribute_names" \
     --expression-attribute-values file://"$attribute_values" \
     --projection-expression "$projection_expression")
 fi

 local error_code=${?}

 if [[ $error_code -ne 0 ]]; then
   aws_cli_error_log $error_code
   errecho "ERROR: AWS reports query operation failed.$response"
   return 1
 fi
```

```
  echo "$response"

  return 0
}

###############################################################################
# function dynamodb_scan
#
# This function scans a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -f filter_expression  -- The filter expression.
#       -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names.
#       -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_scan() {
  local table_name filter_expression expression_attribute_names
 expression_attribute_values projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_scan"
    echo "Scan a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -f filter_expression  -- The filter expression."
    echo " -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names."
    echo " -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
```

```
 }

 while getopts "n:f:a:v:p:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     f) filter_expression="${OPTARG}" ;;
     a) expression_attribute_names="${OPTARG}" ;;
     v) expression_attribute_values="${OPTARG}" ;;
     p) projection_expression="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$filter_expression" ]]; then
   errecho "ERROR: You must provide a filter expression with the -f parameter."
   usage
   return 1
 fi

 if [[ -z "$expression_attribute_names" ]]; then
   errecho "ERROR: You must provide expression attribute names with the -a
parameter."
   usage
   return 1
 fi

 if [[ -z "$expression_attribute_values" ]]; then
   errecho "ERROR: You must provide expression attribute values with the -v
parameter."
```

```
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values")
  else
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports scan operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}

###############################################################################
# function dynamodb_delete_item
#
# This function deletes an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to delete.
#
#  Returns:
#       0 - If successful.
```

```
#       1 - If it fails.
###############################################################################
function dynamodb_delete_item() {
  local table_name keys response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_delete_item"
    echo "Delete an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -k keys  -- Path to json file containing the keys that identify the
 item to delete."
    echo ""
  }
  while getopts "n:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
```

```
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho ""

  response=$(aws dynamodb delete-item \
    --table-name "$table_name" \
    --key file://"$keys")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-item operation failed.$response"
    return 1
  fi

  return 0

}

###############################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function dynamodb_delete_table"
    echo "Deletes an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to delete."
```

```
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho ""

  response=$(aws dynamodb delete-table \
    --table-name "$table_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-table operation failed.$response"
    return 1
  fi

  return 0
}
```

Las funciones de utilidad usadas en este escenario.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
```

```
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de comandos de AWS CLI.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Examen
  - UpdateItem

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    {
        Aws::Client::ClientConfiguration clientConfig;
        //  1. Create a table with partition: year (N) and sort: title (S).
 (CreateTable)
        if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

            AwsDoc::DynamoDB::dynamodbGettingStartedScenario(clientConfig);

            // 9. Delete the table. (DeleteTable)
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
        }
    }

//! Scenario to modify and query a DynamoDB table.
/*!
  \sa dynamodbGettingStartedScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::dynamodbGettingStartedScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;
    std::cout << "Welcome to the Amazon DynamoDB getting started demo." <<
 std::endl;
    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;

    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // 2. Add a new movie.
    Aws::String title;
```

```
   float rating;
   int year;
   Aws::String plot;
   {
       title = askQuestion(
               "Enter the title of a movie you want to add to the table: ");
       year = askQuestionForInt("What year was it released? ");
       rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
it? ",
                                         1, 10);
       plot = askQuestion("Summarize the plot for me: ");

       Aws::DynamoDB::Model::PutItemRequest putItemRequest;
       putItemRequest.SetTableName(MOVIE_TABLE_NAME);

       putItemRequest.AddItem(YEAR_KEY,

Aws::DynamoDB::Model::AttributeValue().SetN(year));
       putItemRequest.AddItem(TITLE_KEY,

Aws::DynamoDB::Model::AttributeValue().SetS(title));

       // Create attribute for the info map.
       Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

       std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
               ALLOCATION_TAG.c_str());
       ratingAttribute->SetN(rating);
       infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

       std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
               ALLOCATION_TAG.c_str());
       plotAttribute->SetS(plot);
       infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);

       putItemRequest.AddItem(INFO_KEY, infoMapAttribute);

       Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
               putItemRequest);
       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to add an item: " <<
outcome.GetError().GetMessage()
```

```
                    << std::endl;
            return false;
        }
    }

    std::cout << "\nAdded '" << title << "' to '" << MOVIE_TABLE_NAME << "'."
              << std::endl;

    // 3. Update the rating and plot of the movie by using an update expression.
    {
        rating = askQuestionForFloatRange(
                Aws::String("\nLet's update your movie.\nYou rated it  ") +
                std::to_string(rating)
                + ", what new rating would you give it? ", 1, 10);
        plot = askQuestion(Aws::String("You summarized the plot as '") + plot +
                           "'.\nWhat would you say now? ");

        Aws::DynamoDB::Model::UpdateItemRequest request;
        request.SetTableName(MOVIE_TABLE_NAME);
        request.AddKey(TITLE_KEY,
Aws::DynamoDB::Model::AttributeValue().SetS(title));
        request.AddKey(YEAR_KEY,
Aws::DynamoDB::Model::AttributeValue().SetN(year));
        std::stringstream expressionStream;
        expressionStream << "set " << INFO_KEY << "." << RATING_KEY << " =:r, "
                         << INFO_KEY << "." << PLOT_KEY << " =:p";
        request.SetUpdateExpression(expressionStream.str());
        request.SetExpressionAttributeValues({
                                                     {":r",
Aws::DynamoDB::Model::AttributeValue().SetN(
                                                             rating)},
                                                     {":p",
Aws::DynamoDB::Model::AttributeValue().SetS(
                                                             plot)}
                                             });

        request.SetReturnValues(Aws::DynamoDB::Model::ReturnValue::UPDATED_NEW);

        const Aws::DynamoDB::Model::UpdateItemOutcome &result =
dynamoClient.UpdateItem(
                request);
        if (!result.IsSuccess()) {
            std::cerr << "Error updating movie " + result.GetError().GetMessage()
                      << std::endl;
```

```
            return false;
        }
    }

    std::cout << "\nUpdated '" << title << "' with new attributes:" << std::endl;

    // 4. Put 250 movies in the table from moviedata.json.
    {
        std::cout << "Adding movies from a json file to the database." <<
std::endl;
        const size_t MAX_SIZE_FOR_BATCH_WRITE = 25;
        const size_t MOVIES_TO_WRITE = 10 * MAX_SIZE_FOR_BATCH_WRITE;
        Aws::String jsonString = getMovieJSON();
        if (!jsonString.empty()) {
            Aws::Utils::Json::JsonValue json(jsonString);
            Aws::Utils::Array<Aws::Utils::Json::JsonView> movieJsons =
json.View().AsArray();
            Aws::Vector<Aws::DynamoDB::Model::WriteRequest> writeRequests;

            // To add movies with a cross-section of years, use an appropriate
increment
            // value for iterating through the database.
            size_t increment = movieJsons.GetLength() / MOVIES_TO_WRITE;
            for (size_t i = 0; i < movieJsons.GetLength(); i += increment) {
                writeRequests.push_back(Aws::DynamoDB::Model::WriteRequest());
                Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
putItems = movieJsonViewToAttributeMap(
                        movieJsons[i]);
                Aws::DynamoDB::Model::PutRequest putRequest;
                putRequest.SetItem(putItems);
                writeRequests.back().SetPutRequest(putRequest);
                if (writeRequests.size() == MAX_SIZE_FOR_BATCH_WRITE) {
                    Aws::DynamoDB::Model::BatchWriteItemRequest request;
                    request.AddRequestItems(MOVIE_TABLE_NAME, writeRequests);
                    const Aws::DynamoDB::Model::BatchWriteItemOutcome &outcome =
dynamoClient.BatchWriteItem(
                            request);
                    if (!outcome.IsSuccess()) {
                        std::cerr << "Unable to batch write movie data: "
                                  << outcome.GetError().GetMessage()
                                  << std::endl;
                        writeRequests.clear();
                        break;
                    }
```

```
                    else {
                        std::cout << "Added batch of " << writeRequests.size()
                                  << " movies to the database."
                                  << std::endl;
                    }
                    writeRequests.clear();
                }
            }
        }

    std::cout << std::setfill('*') << std::setw(ASTERISK_FILL_WIDTH) << " "
              << std::endl;

    // 5. Get a movie by Key (partition + sort).
    {
        Aws::String titleToGet("King Kong");
        Aws::String answer = askQuestion(Aws::String(
                "Let's move on...Would you like to get info about '" + titleToGet
+
                "'? (y/n) "));
        if (answer == "y") {
            Aws::DynamoDB::Model::GetItemRequest request;
            request.SetTableName(MOVIE_TABLE_NAME);
            request.AddKey(TITLE_KEY,

Aws::DynamoDB::Model::AttributeValue().SetS(titleToGet));
            request.AddKey(YEAR_KEY,
Aws::DynamoDB::Model::AttributeValue().SetN(1933));

            const Aws::DynamoDB::Model::GetItemOutcome &result =
dynamoClient.GetItem(
                    request);
            if (!result.IsSuccess()) {
                std::cerr << "Error " << result.GetError().GetMessage();
            }
            else {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = result.GetResult().GetItem();
                if (!item.empty()) {
                    std::cout << "\nHere's what I found:" << std::endl;
                    printMovieInfo(item);
                }
                else {
```

```
                    std::cout << "\nThe movie was not found in the database."
                              << std::endl;
                }
            }
        }
    }

    // 6. Use Query with a key condition expression to return all movies
    //    released in a given year.
    Aws::String doAgain = "n";
    do {
        Aws::DynamoDB::Model::QueryRequest req;

        req.SetTableName(MOVIE_TABLE_NAME);

        // "year" is a DynamoDB reserved keyword and must be replaced with an
        // expression attribute name.
        req.SetKeyConditionExpression("#dynobase_year = :valueToMatch");
        req.SetExpressionAttributeNames({{"#dynobase_year", YEAR_KEY}});

        int yearToMatch = askQuestionForIntRange(
                "\nLet's get a list of movies released in"
                " a given year. Enter a year between 1972 and 2018 ",
                1972, 2018);
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
attributeValues;
        attributeValues.emplace(":valueToMatch",
                                Aws::DynamoDB::Model::AttributeValue().SetN(
                                        yearToMatch));
        req.SetExpressionAttributeValues(attributeValues);

        const Aws::DynamoDB::Model::QueryOutcome &result =
dynamoClient.Query(req);
        if (result.IsSuccess()) {
            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetResult().GetItems();
            if (!items.empty()) {
                std::cout << "\nThere were " << items.size()
                          << " movies in the database from "
                          << yearToMatch << "." << std::endl;
                for (const auto &item: items) {
                    printMovieInfo(item);
                }
                doAgain = "n";
```

```
            }
            else {
                std::cout << "\nNo movies from " << yearToMatch
                          << " were found in the database"
                          << std::endl;
                doAgain = askQuestion(Aws::String("Try another year? (y/n) "));
            }
        }
        else {
            std::cerr << "Failed to Query items: " <<
result.GetError().GetMessage()
                      << std::endl;
        }

    } while (doAgain == "y");

    //  7. Use Scan to return movies released within a range of years.
    //     Show how to paginate data using ExclusiveStartKey. (Scan +
FilterExpression)
    {
        int startYear = askQuestionForIntRange("\nNow let's scan a range of years
"
                                               "for movies in the database. Enter
a start year: ",
                                               1972, 2018);
        int endYear = askQuestionForIntRange("\nEnter an end year: ",
                                             startYear, 2018);
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
exclusiveStartKey;
        do {
            Aws::DynamoDB::Model::ScanRequest scanRequest;
            scanRequest.SetTableName(MOVIE_TABLE_NAME);
            scanRequest.SetFilterExpression(
                    "#dynobase_year >= :startYear AND #dynobase_year
<= :endYear");
            scanRequest.SetExpressionAttributeNames({{"#dynobase_year",
YEAR_KEY}});

            Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
attributeValues;
            attributeValues.emplace(":startYear",
                                    Aws::DynamoDB::Model::AttributeValue().SetN(
                                            startYear));
            attributeValues.emplace(":endYear",
```

```
                                    Aws::DynamoDB::Model::AttributeValue().SetN(
                                            endYear));
            scanRequest.SetExpressionAttributeValues(attributeValues);

            if (!exclusiveStartKey.empty()) {
                scanRequest.SetExclusiveStartKey(exclusiveStartKey);
            }

            const Aws::DynamoDB::Model::ScanOutcome &result = dynamoClient.Scan(
                    scanRequest);
            if (result.IsSuccess()) {
                const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetResult().GetItems();
                if (!items.empty()) {
                    std::stringstream stringStream;
                    stringStream << "\nFound " << items.size() << " movies in one
scan."
                                 << " How many would you like to see? ";
                    size_t count = askQuestionForInt(stringStream.str());
                    for (size_t i = 0; i < count && i < items.size(); ++i) {
                        printMovieInfo(items[i]);
                    }
                }
                else {
                    std::cout << "\nNo movies in the database between " <<
startYear <<
                              " and " << endYear << "." << std::endl;
                }

                exclusiveStartKey = result.GetResult().GetLastEvaluatedKey();
                if (!exclusiveStartKey.empty()) {
                    std::cout << "Not all movies were retrieved. Scanning for
more."
                              << std::endl;
                }
                else {
                    std::cout << "All movies were retrieved with this scan."
                              << std::endl;
                }
            }
            else {
                std::cerr << "Failed to Scan movies: "
                          << result.GetError().GetMessage() << std::endl;
            }
```

```
        } while (!exclusiveStartKey.empty());
    }

    // 8. Delete a movie. (DeleteItem)
    {
        std::stringstream stringStream;
        stringStream << "\nWould you like to delete the movie " << title
                     << " from the database? (y/n) ";
        Aws::String answer = askQuestion(stringStream.str());
        if (answer == "y") {
            Aws::DynamoDB::Model::DeleteItemRequest request;
            request.AddKey(YEAR_KEY,
 Aws::DynamoDB::Model::AttributeValue().SetN(year));
            request.AddKey(TITLE_KEY,
                           Aws::DynamoDB::Model::AttributeValue().SetS(title));
            request.SetTableName(MOVIE_TABLE_NAME);

            const Aws::DynamoDB::Model::DeleteItemOutcome &result =
 dynamoClient.DeleteItem(
                    request);
            if (result.IsSuccess()) {
                std::cout << "\nRemoved \"" << title << "\" from the database."
                          << std::endl;
            }
            else {
                std::cerr << "Failed to delete the movie: "
                          << result.GetError().GetMessage()
                          << std::endl;
            }
        }
    }

    return true;
}

//! Routine to convert a JsonView object to an attribute map.
/*!
  \sa movieJsonViewToAttributeMap()
  \param jsonView: Json view object.
  \return map: Map that can be used in a DynamoDB request.
 */
Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
AwsDoc::DynamoDB::movieJsonViewToAttributeMap(
        const Aws::Utils::Json::JsonView &jsonView) {
```

```
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> result;

    if (jsonView.KeyExists(YEAR_KEY)) {
        result[YEAR_KEY].SetN(jsonView.GetInteger(YEAR_KEY));
    }
    if (jsonView.KeyExists(TITLE_KEY)) {
        result[TITLE_KEY].SetS(jsonView.GetString(TITLE_KEY));
    }
    if (jsonView.KeyExists(INFO_KEY)) {
        Aws::Map<Aws::String, const
 std::shared_ptr<Aws::DynamoDB::Model::AttributeValue>> infoMap;
        Aws::Utils::Json::JsonView infoView = jsonView.GetObject(INFO_KEY);
        if (infoView.KeyExists(RATING_KEY)) {
            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> attributeValue
 = std::make_shared<Aws::DynamoDB::Model::AttributeValue>();
            attributeValue->SetN(infoView.GetDouble(RATING_KEY));
            infoMap.emplace(std::make_pair(RATING_KEY, attributeValue));
        }
        if (infoView.KeyExists(PLOT_KEY)) {
            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> attributeValue
 = std::make_shared<Aws::DynamoDB::Model::AttributeValue>();
            attributeValue->SetS(infoView.GetString(PLOT_KEY));
            infoMap.emplace(std::make_pair(PLOT_KEY, attributeValue));
        }

        result[INFO_KEY].SetM(infoMap);
    }

    return result;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;

    {
```

```
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
                std::cerr << "Failed to create table: "
```

```
                          << result.GetError().GetMessage();
                return false;
            }
        }
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
 clientConfiguration)) {
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
```

```
                  << std::endl;
    }

    return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
```

```
    }
    return false;
}
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK para C++.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Examen
  - UpdateItem

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Ejecutar un escenario interactivo para crear la tabla y realizar acciones en ella.

```
import (
 "context"
 "fmt"
 "log"
 "strings"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunMovieScenario is an interactive example that shows you how to use the AWS
 SDK for Go
// to create and use an Amazon DynamoDB table that stores data about movies.
//
//  1. Create a table that can hold movie data.
//  2. Put, get, and update a single movie in the table.
//  3. Write movie data to the table from a sample JSON file.
//  4. Query for movies that were released in a given year.
//  5. Scan for movies that were released in a range of years.
//  6. Delete a movie from the table.
//  7. Delete the table.
//
// This example creates a DynamoDB service client from the specified sdkConfig so
 that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
//
// The specified movie sampler is used to get sample data from a URL that is
 loaded
// into the named table.
func RunMovieScenario(
 ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner,
 tableName string,
 movieSampler actions.IMovieSampler) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB getting started demo.")
 log.Println(strings.Repeat("-", 88))

 tableBasics := actions.TableBasics{TableName: tableName,
```

```
 DynamoDbClient: dynamodb.NewFromConfig(sdkConfig)}

exists, err := tableBasics.TableExists(ctx)
if err != nil {
 panic(err)
}
if !exists {
 log.Printf("Creating table %v...\n", tableName)
 _, err = tableBasics.CreateMovieTable(ctx)
 if err != nil {
  panic(err)
 } else {
  log.Printf("Created table %v.\n", tableName)
 }
} else {
 log.Printf("Table %v already exists.\n", tableName)
}

var customMovie actions.Movie
customMovie.Title = questioner.Ask("Enter a movie title to add to the table:",
 demotools.NotEmpty{})
customMovie.Year = questioner.AskInt("What year was it released?",
 demotools.NotEmpty{}, demotools.InIntRange{Lower: 1900, Upper: 2030})
customMovie.Info = map[string]interface{}{}
customMovie.Info["rating"] = questioner.AskFloat64(
 "Enter a rating between 1 and 10:",
 demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10})
customMovie.Info["plot"] = questioner.Ask("What's the plot? ",
 demotools.NotEmpty{})
err = tableBasics.AddMovie(ctx, customMovie)
if err == nil {
 log.Printf("Added %v to the movie table.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Let's update your movie. You previously rated it %v.\n",
customMovie.Info["rating"])
customMovie.Info["rating"] = questioner.AskFloat64(
 "What new rating would you give it?",
 demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10})
log.Printf("You summarized the plot as '%v'.\n", customMovie.Info["plot"])
customMovie.Info["plot"] = questioner.Ask("What would you say now?",
 demotools.NotEmpty{})
attributes, err := tableBasics.UpdateMovie(ctx, customMovie)
```

```
if err == nil {
 log.Printf("Updated %v with new values.\n", customMovie.Title)
 for _, attVal := range attributes {
  for valKey, val := range attVal {
   log.Printf("\t%v: %v\n", valKey, val)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Printf("Getting movie data from %v and adding 250 movies to the table...\n",
 movieSampler.GetURL())
movies := movieSampler.GetSampleMovies()
written, err := tableBasics.AddMovieBatch(ctx, movies, 250)
if err != nil {
 panic(err)
} else {
 log.Printf("Added %v movies to the table.\n", written)
}

show := 10
if show > written {
 show = written
}
log.Printf("The first %v movies in the table are:", show)
for index, movie := range movies[:show] {
 log.Printf("\t%v. %v\n", index+1, movie.Title)
}
movieIndex := questioner.AskInt(
 "Enter the number of a movie to get info about it: ",
 demotools.InIntRange{Lower: 1, Upper: show},
)
movie, err := tableBasics.GetMovie(ctx, movies[movieIndex-1].Title,
movies[movieIndex-1].Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))

log.Println("Let's get a list of movies released in a given year.")
releaseYear := questioner.AskInt("Enter a year between 1972 and 2018: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018},
)
releases, err := tableBasics.Query(ctx, releaseYear)
```

```
if err == nil {
 if len(releases) == 0 {
  log.Printf("I couldn't find any movies released in %v!\n", releaseYear)
 } else {
  for _, movie = range releases {
   log.Println(movie)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Println("Now let's scan for movies released in a range of years.")
startYear := questioner.AskInt("Enter a year: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018})
endYear := questioner.AskInt("Enter another year: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018})
releases, err = tableBasics.Scan(ctx, startYear, endYear)
if err == nil {
 if len(releases) == 0 {
  log.Printf("I couldn't find any movies released between %v and %v!\n",
startYear, endYear)
 } else {
  log.Printf("Found %v movies. In this list, the plot is <nil> because "+
   "we used a projection expression when scanning for items to return only "+
   "the title, year, and rating.\n", len(releases))
  for _, movie = range releases {
   log.Println(movie)
  }
 }
}
log.Println(strings.Repeat("-", 88))

var tables []string
if questioner.AskBool("Do you want to list all of your tables? (y/n) ", "y") {
 tables, err = tableBasics.ListTables(ctx)
 if err == nil {
  log.Printf("Found %v tables:", len(tables))
  for _, table := range tables {
   log.Printf("\t%v", table)
  }
 }
}
log.Println(strings.Repeat("-", 88))
```

```
 log.Printf("Let's remove your movie '%v'.\n", customMovie.Title)
 if questioner.AskBool("Do you want to delete it from the table? (y/n) ", "y") {
  err = tableBasics.DeleteMovie(ctx, customMovie)
 }
 if err == nil {
  log.Printf("Deleted %v.\n", customMovie.Title)
 }

 if questioner.AskBool("Delete the table, too? (y/n)", "y") {
  err = tableBasics.DeleteTable(ctx)
 } else {
  log.Println("Don't forget to delete the table when you're done or you might " +
   "incur charges on your account.")
 }
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Defina una estructura Película para utilizar en este ejemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
```

```
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Crear una estructura y los métodos que llaman a las acciones de DynamoDB.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
```

```
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// TableExists determines whether a DynamoDB table exists.
func (basics TableBasics) TableExists(ctx context.Context) (bool, error) {
 exists := true
 _, err := basics.DynamoDbClient.DescribeTable(
  ctx, &dynamodb.DescribeTableInput{TableName: aws.String(basics.TableName)},
 )
 if err != nil {
  var notFoundEx *types.ResourceNotFoundException
  if errors.As(err, &notFoundEx) {
   log.Printf("Table %v does not exist.\n", basics.TableName)
   err = nil
  } else {
   log.Printf("Couldn't determine existence of table %v. Here's why: %v\n",
 basics.TableName, err)
  }
  exists = false
 }
 return exists, err
}



// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
```

```
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}



// ListTables lists the DynamoDB table names for the current account.
func (basics TableBasics) ListTables(ctx context.Context) ([]string, error) {
 var tableNames []string
 var output *dynamodb.ListTablesOutput
 var err error
```

```
 tablePaginator := dynamodb.NewListTablesPaginator(basics.DynamoDbClient,
 &dynamodb.ListTablesInput{})
 for tablePaginator.HasMorePages() {
  output, err = tablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't list tables. Here's why: %v\n", err)
   break
  } else {
   tableNames = append(tableNames, output.TableNames...)
  }
 }
 return tableNames, err
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
```

```
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx,
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}



// AddMovieBatch adds a slice of movies to the DynamoDB table. The function sends
// batches of 25 movies to DynamoDB until all movies are added or it reaches the
// specified maximum.
func (basics TableBasics) AddMovieBatch(ctx context.Context, movies []Movie,
 maxMovies int) (int, error) {
 var err error
 var item map[string]types.AttributeValue
 written := 0
 batchSize := 25 // DynamoDB allows a maximum batch size of 25 items.
 start := 0
 end := start + batchSize
 for start < maxMovies && start < len(movies) {
  var writeReqs []types.WriteRequest
  if end > len(movies) {
   end = len(movies)
  }
```

```
  for _, movie := range movies[start:end] {
   item, err = attributevalue.MarshalMap(movie)
   if err != nil {
    log.Printf("Couldn't marshal movie %v for batch writing. Here's why: %v\n",
 movie.Title, err)
   } else {
    writeReqs = append(
     writeReqs,
     types.WriteRequest{PutRequest: &types.PutRequest{Item: item}},
    )
   }
  }
  _, err = basics.DynamoDbClient.BatchWriteItem(ctx,
 &dynamodb.BatchWriteItemInput{
   RequestItems: map[string][]types.WriteRequest{basics.TableName: writeReqs}})
  if err != nil {
   log.Printf("Couldn't add a batch of movies to %v. Here's why: %v\n",
 basics.TableName, err)
  } else {
   written += len(writeReqs)
  }
  start = end
  end += batchSize
 }

 return written, err
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
```

```
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
```

```
    }
   }
  }
 }
 return movies, err
}



// Scan gets all movies in the DynamoDB table that were released in a range of
 years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
// expressions.
func (basics TableBasics) Scan(ctx context.Context, startYear int, endYear int)
 ([]Movie, error) {
 var movies []Movie
 var err error
 var response *dynamodb.ScanOutput
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"),
 expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why:
 %v\n",
     startYear, endYear, err)
    break
```

```
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}



// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(ctx context.Context, movie Movie) error {
 _, err := basics.DynamoDbClient.DeleteItem(ctx, &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
 return err
}
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK para Go.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Examen
  - UpdateItem

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear una tabla de DynamoDB.

```
// Create a table with a Sort key.
public static void createTable(DynamoDbClient ddb, String tableName) {
    DynamoDbWaiter dbWaiter = ddb.waiter();
    ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

    // Define attributes.
    attributeDefinitions.add(AttributeDefinition.builder()
        .attributeName("year")
        .attributeType("N")
        .build());

    attributeDefinitions.add(AttributeDefinition.builder()
```

```
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
            .attributeName("title")
            .keyType(KeyType.RANGE)
            .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
```

```
    }
```

Crear una función auxiliar para descargar y extraer el archivo JSON de muestra.

```
   // Load data into the table.
   public static void loadData(DynamoDbClient ddb, String tableName, String
fileName) throws IOException {
       DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
           .dynamoDbClient(ddb)
           .build();

       DynamoDbTable<Movies> mappedTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
       JsonParser parser = new JsonFactory().createParser(new File(fileName));
       com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
       Iterator<JsonNode> iter = rootNode.iterator();
       ObjectNode currentNode;
       int t = 0;
       while (iter.hasNext()) {
           // Only add 200 Movies to the table.
           if (t == 200)
               break;
           currentNode = (ObjectNode) iter.next();

           int year = currentNode.path("year").asInt();
           String title = currentNode.path("title").asText();
           String info = currentNode.path("info").toString();

           Movies movies = new Movies();
           movies.setYear(year);
           movies.setTitle(title);
           movies.setInfo(info);

           // Put the data into the Amazon DynamoDB Movie table.
           mappedTable.putItem(movies);
           t++;
       }
   }
```

Obtener un elemento de una tabla.

```
    public static void getItem(DynamoDbClient ddb) {

        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put("year", AttributeValue.builder()
            .n("1933")
            .build());

        keyToGet.put("title", AttributeValue.builder()
            .s("King Kong")
            .build());

        GetItemRequest request = GetItemRequest.builder()
            .key(keyToGet)
            .tableName("Movies")
            .build();

        try {
            Map<String, AttributeValue> returnedItem =
ddb.getItem(request).item();

            if (returnedItem != null) {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");

                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
returnedItem.get(key1).toString());
                }
            } else {
                System.out.format("No item found with the key %s!\n", "year");
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

Ejemplo completo.

```
/**
```

```
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 * <p>
 * This Java example performs these tasks:
 * <p>
 * 1. Creates the Amazon DynamoDB Movie table with partition and sort key.
 * 2. Puts data into the Amazon DynamoDB table from a JSON document using the
 * Enhanced client.
 * 3. Gets data from the Movie table.
 * 4. Adds a new item.
 * 5. Updates an item.
 * 6. Uses a Scan to query items using the Enhanced client.
 * 7. Queries all items where the year is 2013 using the Enhanced Client.
 * 8. Deletes the table.
 */

public class Scenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws IOException {
        String tableName = "Movies";
        String fileName = "../../../resources/sample_files/movies.json";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon DynamoDB example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println(
            "1. Creating an Amazon DynamoDB table named Movies with a key named
 year and a sort key named title.");
        createTable(ddb, tableName);
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("2. Loading data into the Amazon DynamoDB table.");
        loadData(ddb, tableName, fileName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Getting data from the Movie table.");
        getItem(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Putting a record into the Amazon DynamoDB
table.");
        putRecord(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Updating a record.");
        updateTableItem(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Scanning the Amazon DynamoDB table.");
        scanMovies(ddb, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Querying the Movies released in 2013.");
        queryTable(ddb);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Deleting the Amazon DynamoDB table.");
        deleteDynamoDBTable(ddb, tableName);
        System.out.println(DASHES);

        ddb.close();
    }

    // Create a table with a Sort key.
    public static void createTable(DynamoDbClient ddb, String tableName) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();
```

```
        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("year")
            .attributeType("N")
            .build());

        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
            .attributeName("title")
            .keyType(KeyType.RANGE)
            .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
```

```
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    // Query the table.
    public static void queryTable(DynamoDbClient ddb) {
        try {
            DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

            DynamoDbTable<Movies> custTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
            QueryConditional queryConditional = QueryConditional
                .keyEqualTo(Key.builder()
                    .partitionValue(2013)
                    .build());

            // Get items in the table and write out the ID value.
            Iterator<Movies> results =
custTable.query(queryConditional).items().iterator();
            String result = "";

            while (results.hasNext()) {
                Movies rec = results.next();
                System.out.println("The title of the movie is " +
rec.getTitle());
                System.out.println("The movie information  is " + rec.getInfo());
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    // Scan the table.
    public static void scanMovies(DynamoDbClient ddb, String tableName) {
```

```
        System.out.println("******* Scanning all movies.\n");
        try {
            DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

            DynamoDbTable<Movies> custTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
            Iterator<Movies> results = custTable.scan().items().iterator();
            while (results.hasNext()) {
                Movies rec = results.next();
                System.out.println("The movie title is " + rec.getTitle());
                System.out.println("The movie year is " + rec.getYear());
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    // Load data into the table.
    public static void loadData(DynamoDbClient ddb, String tableName, String
fileName) throws IOException {
        DynamoDbEnhancedClient enhancedClient = DynamoDbEnhancedClient.builder()
            .dynamoDbClient(ddb)
            .build();

        DynamoDbTable<Movies> mappedTable = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));
        JsonParser parser = new JsonFactory().createParser(new File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        Iterator<JsonNode> iter = rootNode.iterator();
        ObjectNode currentNode;
        int t = 0;
        while (iter.hasNext()) {
            // Only add 200 Movies to the table.
            if (t == 200)
                break;
            currentNode = (ObjectNode) iter.next();

            int year = currentNode.path("year").asInt();
```

```
            String title = currentNode.path("title").asText();
            String info = currentNode.path("info").toString();

            Movies movies = new Movies();
            movies.setYear(year);
            movies.setTitle(title);
            movies.setInfo(info);

            // Put the data into the Amazon DynamoDB Movie table.
            mappedTable.putItem(movies);
            t++;
        }
    }

    // Update the record to include show only directors.
    public static void updateTableItem(DynamoDbClient ddb, String tableName) {
        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put("year", AttributeValue.builder().n("1933").build());
        itemKey.put("title", AttributeValue.builder().s("King Kong").build());

        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put("info", AttributeValueUpdate.builder()
            .value(AttributeValue.builder().s("{\"directors\":[\"Merian C. Cooper
\",\"Ernest B. Schoedsack\"]")
                .build())
            .action(AttributeAction.PUT)
            .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(itemKey)
            .attributeUpdates(updatedValues)
            .build();

        try {
            ddb.updateItem(request);
        } catch (ResourceNotFoundException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
```

```
        System.out.println("Item was updated!");
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
{
        DeleteTableRequest request = DeleteTableRequest.builder()
            .tableName(tableName)
            .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }

    public static void putRecord(DynamoDbClient ddb) {
        try {
            DynamoDbEnhancedClient enhancedClient =
DynamoDbEnhancedClient.builder()
                .dynamoDbClient(ddb)
                .build();

            DynamoDbTable<Movies> table = enhancedClient.table("Movies",
TableSchema.fromBean(Movies.class));

            // Populate the Table.
            Movies record = new Movies();
            record.setYear(2020);
            record.setTitle("My Movie2");
            record.setInfo("no info");
            table.putItem(record);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Added a new movie to the table.");
    }

    public static void getItem(DynamoDbClient ddb) {
```

```
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put("year", AttributeValue.builder()
            .n("1933")
            .build());

        keyToGet.put("title", AttributeValue.builder()
            .s("King Kong")
            .build());

        GetItemRequest request = GetItemRequest.builder()
            .key(keyToGet)
            .tableName("Movies")
            .build();

        try {
            Map<String, AttributeValue> returnedItem =
 ddb.getItem(request).item();

            if (returnedItem != null) {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");

                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            } else {
                System.out.format("No item found with the key %s!\n", "year");
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK for Java 2.x.
  - BatchWriteItem

- CreateTable
- DeleteItem
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Query
- Examen
- UpdateItem

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import { readFileSync } from "node:fs";
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";

/**
 * This module is a convenience library. It abstracts Amazon DynamoDB's data type
 * descriptors (such as S, N, B, and BOOL) by marshalling JavaScript objects into
 * AttributeValue shapes.
 */
import {
  BatchWriteCommand,
  DeleteCommand,
  DynamoDBDocumentClient,
```

```
  GetCommand,
  PutCommand,
  UpdateCommand,
  paginateQuery,
  paginateScan,
} from "@aws-sdk/lib-dynamodb";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);
const tableName = getUniqueName("Movies");
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);

export const main = async () => {
  /**
   * Create a table.
   */

  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "year",
        // 'N' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "N",
      },
      { AttributeName: "title", AttributeType: "S" },
    ],
```

```
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [
      // The way your data is accessed determines how you structure your keys.
      // The movies table will be queried for movies by year. It makes sense
      // to make year our partition (HASH) key.
      { AttributeName: "year", KeyType: "HASH" },
      { AttributeName: "title", KeyType: "RANGE" },
    ],
  });

  log("Creating a table.");
  const createTableResponse = await client.send(createTableCommand);
  log(`Table created: ${JSON.stringify(createTableResponse.TableDescription)}`);

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Add a movie to the table.
   */

  log("Adding a single movie to the table.");
  // PutCommand is the first example usage of 'lib-dynamodb'.
  const putCommand = new PutCommand({
    TableName: tableName,
    Item: {
      // In 'client-dynamodb', the AttributeValue would be required (`year: { N:
 1981 }`)
      // 'lib-dynamodb' simplifies the usage ( `year: 1981` )
      year: 1981,
      // The preceding KeySchema defines 'title' as our sort (RANGE) key, so
 'title'
      // is required.
      title: "The Evil Dead",
      // Every other attribute is optional.
      info: {
        genres: ["Horror"],
```

```
      },
    },
  });
  await docClient.send(putCommand);
  log("The movie was added.");

  /**
   * Get a movie from the table.
   */

  log("Getting a single movie from the table.");
  const getCommand = new GetCommand({
    TableName: tableName,
    // Requires the complete primary key. For the movies table, the primary key
    // is only the id (partition key).
    Key: {
      year: 1981,
      title: "The Evil Dead",
    },
    // Set this to make sure that recent writes are reflected.
    // For more information, see https://docs.aws.amazon.com/amazondynamodb/
latest/developerguide/HowItWorks.ReadConsistency.html.
    ConsistentRead: true,
  });
  const getResponse = await docClient.send(getCommand);
  log(`Got the movie: ${JSON.stringify(getResponse.Item)}`);

  /**
   * Update a movie in the table.
   */

  log("Updating a single movie in the table.");
  const updateCommand = new UpdateCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
    // This update expression appends "Comedy" to the list of genres.
    // For more information on update expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.UpdateExpressions.html
    UpdateExpression: "set #i.#g = list_append(#i.#g, :vals)",
    ExpressionAttributeNames: { "#i": "info", "#g": "genres" },
    ExpressionAttributeValues: {
      ":vals": ["Comedy"],
    },
```

```
  ReturnValues: "ALL_NEW",
});
const updateResponse = await docClient.send(updateCommand);
log(`Movie updated: ${JSON.stringify(updateResponse.Attributes)}`);

/**
 * Delete a movie from the table.
 */

log("Deleting a single movie from the table.");
const deleteCommand = new DeleteCommand({
  TableName: tableName,
  Key: { year: 1981, title: "The Evil Dead" },
});
await client.send(deleteCommand);
log("Movie deleted.");

/**
 * Upload a batch of movies.
 */

log("Adding movies from local JSON file.");
const file = readFileSync(
  `${dirname}../../../../resources/sample_files/movies.json`,
);
const movies = JSON.parse(file.toString());
// chunkArray is a local convenience function. It takes an array and returns
// a generator function. The generator function yields every N items.
const movieChunks = chunkArray(movies, 25);
// For every chunk of 25 movies, make one BatchWrite request.
for (const chunk of movieChunks) {
  const putRequests = chunk.map((movie) => ({
    PutRequest: {
      Item: movie,
    },
  }));

  const command = new BatchWriteCommand({
    RequestItems: {
      [tableName]: putRequests,
    },
  });

  await docClient.send(command);
```

```
  }
  log("Movies added.");

  /**
   * Query for movies by year.
   */

  log("Querying for all movies from 1981.");
  const paginatedQuery = paginateQuery(
    { client: docClient },
    {
      TableName: tableName,
      //For more information about query expressions, see
      // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Query.html#Query.KeyConditionExpressions
      KeyConditionExpression: "#y = :y",
      // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
      // name by using an expression attribute name.
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y": 1981 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1981 = [];
  for await (const page of paginatedQuery) {
    movies1981.push(...page.Items);
  }
  log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

  /**
   * Scan the table for movies between 1980 and 1990.
   */

  log("Scan for movies released between 1980 and 1990");
  // A 'Scan' operation always reads every item in the table. If your design
 requires
  // the use of 'Scan', consider indexing your table or changing your design.
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
  const paginatedScan = paginateScan(
    { client: docClient },
```

```
    {
      TableName: tableName,
      // Scan uses a filter expression instead of a key condition expression.
 Scan will
      // read the entire table and then apply the filter.
      FilterExpression: "#y between :y1 and :y2",
      ExpressionAttributeNames: { "#y": "year" },
      ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
      ConsistentRead: true,
    },
  );
  /**
   * @type { Record<string, any>[] };
   */
  const movies1980to1990 = [];
  for await (const page of paginatedScan) {
    movies1980to1990.push(...page.Items);
  }
  log(
    `Movies: ${movies1980to1990
      .map((m) => `${m.title} (${m.year})`)
      .join(", ")}`,
  );

  /**
   * Delete the table.
   */

  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  log(`Deleting table ${tableName}.`);
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK para JavaScript.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable

- DescribeTable
- GetItem
- PutItem
- Query
- Examen
- UpdateItem

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear una tabla de DynamoDB.

```
suspend fun createScenarioTable(
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
```

```
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        println("The table was successfully created
 ${response.tableDescription?.tableArn}")
    }
}
```

Crear una función auxiliar para descargar y extraer el archivo JSON de muestra.

```
// Load data into the table.
suspend fun loadData(
    tableName: String,
    fileName: String,
) {
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode

    var t = 0
    while (iter.hasNext()) {
        if (t == 50) {
```

```
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()
        putMovie(tableName, year, title, info)
        t++
    }
}

suspend fun putMovie(
    tableNameVal: String,
    year: Int,
    title: String,
    info: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()
    val strVal = year.toString()
    // Add all content to the table.
    itemValues["year"] = AttributeValue.N(strVal)
    itemValues["title"] = AttributeValue.S(title)
    itemValues["info"] = AttributeValue.S(info)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println("Added $title to the Movie table.")
    }
}
```

Obtener un elemento de una tabla.

```
suspend fun getMovie(
    tableNameVal: String,
    keyName: String,
```

```
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.N(keyVal)
    keyToGet["title"] = AttributeValue.S("King Kong")

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

Ejemplo completo.

```
suspend fun main() {
    val tableName = "Movies"
    val fileName = "../../../resources/sample_files/movies.json"
    val partitionAlias = "#a"

    println("Creating an Amazon DynamoDB table named Movies with a key named id
 and a sort key named title.")
    createScenarioTable(tableName, "year")
    loadData(tableName, fileName)
    getMovie(tableName, "year", "1933")
    scanMovies(tableName)
    val count = queryMovieTable(tableName, "year", partitionAlias)
    println("There are $count Movies released in 2013.")
    deletIssuesTable(tableName)
}

suspend fun createScenarioTable(
    tableNameVal: String,
```

```
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        println("The table was successfully created
 ${response.tableDescription?.tableArn}")
    }
}
```

```
// Load data into the table.
suspend fun loadData(
    tableName: String,
    fileName: String,
) {
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode

    var t = 0
    while (iter.hasNext()) {
        if (t == 50) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()
        putMovie(tableName, year, title, info)
        t++
    }
}

suspend fun putMovie(
    tableNameVal: String,
    year: Int,
    title: String,
    info: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()
    val strVal = year.toString()
    // Add all content to the table.
    itemValues["year"] = AttributeValue.N(strVal)
    itemValues["title"] = AttributeValue.S(title)
    itemValues["info"] = AttributeValue.S(info)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }
```

```
    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println("Added $title to the Movie table.")
    }
}

suspend fun getMovie(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.N(keyVal)
    keyToGet["title"] = AttributeValue.S("King Kong")

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}

suspend fun deletIssuesTable(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

```
suspend fun queryMovieTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = "year"

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.N("2013")

    val request =
        QueryRequest {
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
            this.expressionAttributeValues = attrValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}

suspend fun scanMovies(tableNameVal: String) {
    val request =
        ScanRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.scan(request)
        response.items?.forEach { item ->
            item.keys.forEach { key ->
                println("The key name is $key\n")
                println("The value is ${item[key]}")
            }
        }
    }
}
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Kotlin.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Examen
  - UpdateItem

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Dado que este ejemplo utiliza archivos auxiliares, asegúrese de leer las instrucciones del archivo README.md de los ejemplos de PHP.

```
namespace DynamoDb\Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;
use DynamoDb\DynamoDBService;

use function AwsUtilities\loadMovieData;
use function AwsUtilities\testable_readline;

class GettingStartedWithDynamoDB
```

```
{
    public function run()
    {
        echo("\n");
        echo("---------------------------------------\n");
        print("Welcome to the Amazon DynamoDB getting started demo using PHP!
\n");
        echo("---------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDBService();

        $tableName = "ddb_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
```

```
            ],
            'TableName' => $tableName,
        ]);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $key = [
            'Item' => [
                'title' => [
                    'S' => $movieName,
                ],
                'year' => [
                    'N' => $movieYear,
                ],
            ]
        ];
        $attributes = ["rating" =>
            [
                'AttributeName' => 'rating',
                'AttributeType' => 'N',
                'Value' => $rating,
            ],
            'plot' => [
                'AttributeName' => 'plot',
                'AttributeType' => 'S',
                'Value' => $plot,
            ]
        ];
        $service->updateItemAttributesByKey($tableName, $key, $attributes);
        echo "Movie added and updated.";

        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);
```

```
        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
 {$movie['Item']['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

        $movie = $service->getItemByKey($tableName, $key);
        echo "Ok, you have rated {$movie['Item']['title']['S']} as a
 {$movie['Item']['rating']['N']}\n";

        $service->deleteItemByKey($tableName, $key);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something
 lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
 born:\n";
        $oops = "Oops! There were no movies released in that year (that we know
 of).\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
```

```
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK para PHP.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Examen
  - UpdateItem

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear una clase que encapsula una tabla de DynamoDB.

```
from decimal import Decimal
from io import BytesIO
import json
import logging
import os
from pprint import pprint
import requests
from zipfile import ZipFile
import boto3
from boto3.dynamodb.conditions import Key
from botocore.exceptions import ClientError
from question import Question

logger = logging.getLogger(__name__)

class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
```

```
                "Kelly Preston",
                "John C. Reilly"
            ]
        }
    }
"""

def __init__(self, dyn_resource):
    """
    :param dyn_resource: A Boto3 DynamoDB resource.
    """
    self.dyn_resource = dyn_resource
    # The table variable is set during the scenario in the call to
    # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
    self.table = None


def exists(self, table_name):
    """
    Determines whether a table exists. As a side effect, stores the table in
    a member variable.

    :param table_name: The name of the table to check.
    :return: True when the table exists; otherwise, False.
    """
    try:
        table = self.dyn_resource.Table(table_name)
        table.load()
        exists = True
    except ClientError as err:
        if err.response["Error"]["Code"] == "ResourceNotFoundException":
            exists = False
        else:
            logger.error(
                "Couldn't check for existence of %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
    else:
        self.table = table
    return exists
```

```
    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                BillingMode='PAY_PER_REQUEST',
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table


    def list_tables(self):
        """
        Lists the Amazon DynamoDB tables for the current account.

        :return: The list of tables.
        """
        try:
```

```
            tables = []
            for table in self.dyn_resource.tables.all():
                print(table.name)
                tables.append(table)
        except ClientError as err:
            logger.error(
                "Couldn't list tables. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return tables


    def write_batch(self, movies):
        """
        Fills an Amazon DynamoDB table with the specified data, using the Boto3
        Table.batch_writer() function to put the items in the table.
        Inside the context manager, Table.batch_writer builds a list of
        requests. On exiting the context manager, Table.batch_writer starts
sending
        batches of write requests to Amazon DynamoDB and automatically
        handles chunking, buffering, and retrying.

        :param movies: The data to put in the table. Each item must contain at
least
                       the keys required by the schema that was specified when
the
                       table was created.
        """
        try:
            with self.table.batch_writer() as writer:
                for movie in movies:
                    writer.put_item(Item=movie)
        except ClientError as err:
            logger.error(
                "Couldn't load data into table %s. Here's why: %s: %s",
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

```
    def add_movie(self, title, year, plot, rating):
        """
        Adds a movie to the table.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :param plot: The plot summary of the movie.
        :param rating: The quality rating of the movie.
        """
        try:
            self.table.put_item(
                Item={
                    "year": year,
                    "title": title,
                    "info": {"plot": plot, "rating": Decimal(str(rating))},
                }
            )
        except ClientError as err:
            logger.error(
                "Couldn't add movie %s to table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
```

```
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]


    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
```

```
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Items"]


    def scan_movies(self, year_range):
        """
        Scans for movies that were released in a range of years.
        Uses a projection expression to return a subset of data for each movie.

        :param year_range: The range of years to retrieve.
        :return: The list of movies released in the specified years.
        """
        movies = []
        scan_kwargs = {
            "FilterExpression": Key("year").between(
                year_range["first"], year_range["second"]
            ),
            "ProjectionExpression": "#yr, title, info.rating",
            "ExpressionAttributeNames": {"#yr": "year"},
        }
        try:
            done = False
            start_key = None
            while not done:
                if start_key:
                    scan_kwargs["ExclusiveStartKey"] = start_key
                response = self.table.scan(**scan_kwargs)
                movies.extend(response.get("Items", []))
                start_key = response.get("LastEvaluatedKey", None)
                done = start_key is None
        except ClientError as err:
```

```
            logger.error(
                "Couldn't scan for movies. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

        return movies


    def delete_movie(self, title, year):
        """
        Deletes a movie from the table.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        """
        try:
            self.table.delete_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't delete movie %s. Here's why: %s: %s",
                title,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

Crear una función auxiliar para descargar y extraer el archivo JSON de muestra.

```
def get_sample_movie_data(movie_file_name):
    """
    Gets sample movie data, either from a local file or by first downloading it
 from
    the Amazon DynamoDB developer guide.

    :param movie_file_name: The local file name where the movie data is stored in
 JSON format.
    :return: The movie data as a dict.
    """
    if not os.path.isfile(movie_file_name):
        print(f"Downloading {movie_file_name}...")
        movie_content = requests.get(
            "https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
samples/moviedata.zip"
        )
        movie_zip = ZipFile(BytesIO(movie_content.content))
        movie_zip.extractall()

    try:
        with open(movie_file_name) as movie_file:
            movie_data = json.load(movie_file, parse_float=Decimal)
    except FileNotFoundError:
        print(
            f"File {movie_file_name} not found. You must first download the file
 to "
            "run this demo. See the README for instructions."
        )
        raise
    else:
        # The sample file lists over 4000 movies, return only the first 250.
        return movie_data[:250]
```

Ejecutar un escenario interactivo para crear la tabla y realizar acciones en ella.

```
def run_scenario(table_name, movie_file_name, dyn_resource):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB getting started demo.")
    print("-" * 88)

    movies = Movies(dyn_resource)
    movies_exists = movies.exists(table_name)
    if not movies_exists:
        print(f"\nCreating table {table_name}...")
        movies.create_table(table_name)
        print(f"\nCreated table {movies.table.name}.")

    my_movie = Question.ask_questions(
        [
            Question(
                "title", "Enter the title of a movie you want to add to the
 table: "
            ),
            Question("year", "What year was it released? ", Question.is_int),
            Question(
                "rating",
                "On a scale of 1 - 10, how do you rate it? ",
                Question.is_float,
                Question.in_range(1, 10),
            ),
            Question("plot", "Summarize the plot for me: "),
        ]
    )
    movies.add_movie(**my_movie)
    print(f"\nAdded '{my_movie['title']}' to '{movies.table.name}'.")
    print("-" * 88)

    movie_update = Question.ask_questions(
        [
            Question(
                "rating",
                f"\nLet's update your movie.\nYou rated it {my_movie['rating']},
 what new "
                f"rating would you give it? ",
                Question.is_float,
                Question.in_range(1, 10),
```

```
            ),
            Question(
                "plot",
                f"You summarized the plot as '{my_movie['plot']}'.\nWhat would
you say now? ",
            ),
        ]
    )
    my_movie.update(movie_update)
    updated = movies.update_movie(**my_movie)
    print(f"\nUpdated '{my_movie['title']}' with new attributes:")
    pprint(updated)
    print("-" * 88)

    if not movies_exists:
        movie_data = get_sample_movie_data(movie_file_name)
        print(f"\nReading data from '{movie_file_name}' into your table.")
        movies.write_batch(movie_data)
        print(f"\nWrote {len(movie_data)} movies into {movies.table.name}.")
    print("-" * 88)

    title = "The Lord of the Rings: The Fellowship of the Ring"
    if Question.ask_question(
        f"Let's move on...do you want to get info about '{title}'? (y/n) ",
        Question.is_yesno,
    ):
        movie = movies.get_movie(title, 2001)
        print("\nHere's what I found:")
        pprint(movie)
    print("-" * 88)

    ask_for_year = True
    while ask_for_year:
        release_year = Question.ask_question(
            f"\nLet's get a list of movies released in a given year. Enter a year
between "
            f"1972 and 2018: ",
            Question.is_int,
            Question.in_range(1972, 2018),
        )
        releases = movies.query_movies(release_year)
        if releases:
            print(f"There were {len(releases)} movies released in
{release_year}:")
```

```
            for release in releases:
                print(f"\t{release['title']}")
            ask_for_year = False
        else:
            print(f"I don't know about any movies released in {release_year}!")
            ask_for_year = Question.ask_question(
                "Try another year? (y/n) ", Question.is_yesno
            )
    print("-" * 88)

    years = Question.ask_questions(
        [
            Question(
                "first",
                f"\nNow let's scan for movies released in a range of years. Enter
a year: ",
                Question.is_int,
                Question.in_range(1972, 2018),
            ),
            Question(
                "second",
                "Now enter another year: ",
                Question.is_int,
                Question.in_range(1972, 2018),
            ),
        ]
    )
    releases = movies.scan_movies(years)
    if releases:
        count = Question.ask_question(
            f"\nFound {len(releases)} movies. How many do you want to see? ",
            Question.is_int,
            Question.in_range(1, len(releases)),
        )
        print(f"\nHere are your {count} movies:\n")
        pprint(releases[:count])
    else:
        print(
            f"I don't know about any movies released between {years['first']} "
            f"and {years['second']}."
        )
    print("-" * 88)

    if Question.ask_question(
```

```
        f"\nLet's remove your movie from the table. Do you want to remove "
        f"'{my_movie['title']}'? (y/n)",
        Question.is_yesno,
    ):
        movies.delete_movie(my_movie["title"], my_movie["year"])
        print(f"\nRemoved '{my_movie['title']}' from the table.")
    print("-" * 88)

    if Question.ask_question(f"\nDelete the table? (y/n) ", Question.is_yesno):
        movies.delete_table()
        print(f"Deleted {table_name}.")
    else:
        print(
            "Don't forget to delete the table when you're done or you might incur
 "
            "charges on your account."
        )

    print("\nThanks for watching!")
    print("-" * 88)


if __name__ == "__main__":
    try:
        run_scenario(
            "doc-example-table-movies", "moviedata.json",
 boto3.resource("dynamodb")
        )
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

En este escenario se utiliza la siguiente clase auxiliar para hacer preguntas en un símbolo del sistema.

```
class Question:
    """
    A helper class to ask questions at a command prompt and validate and convert
    the answers.
    """

    def __init__(self, key, question, *validators):
        """
```

```
        :param key: The key that is used for storing the answer in a dict, when
                    multiple questions are asked in a set.
        :param question: The question to ask.
        :param validators: The answer is passed through the list of validators
until
                           one fails or they all pass. Validators may also
convert the
                           answer to another form, such as from a str to an int.
        """
        self.key = key
        self.question = question
        self.validators = Question.non_empty, *validators

    @staticmethod
    def ask_questions(questions):
        """
        Asks a set of questions and stores the answers in a dict.

        :param questions: The list of questions to ask.
        :return: A dict of answers.
        """
        answers = {}
        for question in questions:
            answers[question.key] = Question.ask_question(
                question.question, *question.validators
            )
        return answers

    @staticmethod
    def ask_question(question, *validators):
        """
        Asks a single question and validates it against a list of validators.
        When an answer fails validation, the complaint is printed and the
question
        is asked again.

        :param question: The question to ask.
        :param validators: The list of validators that the answer must pass.
        :return: The answer, converted to its final form by the validators.
        """
        answer = None
        while answer is None:
            answer = input(question)
            for validator in validators:
```

```
                answer, complaint = validator(answer)
                if answer is None:
                    print(complaint)
                    break
        return answer

    @staticmethod
    def non_empty(answer):
        """
        Validates that the answer is not empty.
        :return: The non-empty answer, or None.
        """
        return answer if answer != "" else None, "I need an answer. Please?"

    @staticmethod
    def is_yesno(answer):
        """
        Validates a yes/no answer.
        :return: True when the answer is 'y'; otherwise, False.
        """
        return answer.lower() == "y", ""

    @staticmethod
    def is_int(answer):
        """
        Validates that the answer can be converted to an int.
        :return: The int answer; otherwise, None.
        """
        try:
            int_answer = int(answer)
        except ValueError:
            int_answer = None
        return int_answer, f"{answer} must be a valid integer."

    @staticmethod
    def is_letter(answer):
        """
        Validates that the answer is a letter.
        :return The letter answer, converted to uppercase; otherwise, None.
        """
        return (
            answer.upper() if answer.isalpha() else None,
            f"{answer} must be a single letter.",
        )
```

```
    @staticmethod
    def is_float(answer):
        """
        Validate that the answer can be converted to a float.
        :return The float answer; otherwise, None.
        """
        try:
            float_answer = float(answer)
        except ValueError:
            float_answer = None
        return float_answer, f"{answer} must be a valid float."

    @staticmethod
    def in_range(lower, upper):
        """
        Validate that the answer is within a range. The answer must be of a type
that can
        be compared to the lower and upper bounds.
        :return: The answer, if it is within the range; otherwise, None.
        """

        def _validate(answer):
            return (
                answer if lower <= answer <= upper else None,
                f"{answer} must be between {lower} and {upper}.",
            )

        return _validate
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Python (Boto3).
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem

- PutItem
- Query
- Examen
- UpdateItem

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear una clase que encapsula una tabla de DynamoDB.

```
# Creates an Amazon DynamoDB table that can be used to store movie data.
# The table uses the release year of the movie as the partition key and the
# title as the sort key.
#
# @param table_name [String] The name of the table to create.
# @return [Aws::DynamoDB::Table] The newly created table.
def create_table(table_name)
  @table = @dynamo_resource.create_table(
    table_name: table_name,
    key_schema: [
      { attribute_name: 'year', key_type: 'HASH' }, # Partition key
      { attribute_name: 'title', key_type: 'RANGE' } # Sort key
    ],
    attribute_definitions: [
      { attribute_name: 'year', attribute_type: 'N' },
      { attribute_name: 'title', attribute_type: 'S' }
    ],
    billing_mode: 'PAY_PER_REQUEST'
  )
  @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
  @table
rescue Aws::DynamoDB::Errors::ServiceError => e
  @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
```

```
    raise
  end
```

Crear una función auxiliar para descargar y extraer el archivo JSON de muestra.

```
  # Gets sample movie data, either from a local file or by first downloading it
 from
  # the Amazon DynamoDB Developer Guide.
  #
  # @param movie_file_name [String] The local file name where the movie data is
 stored in JSON format.
  # @return [Hash] The movie data as a Hash.
  def fetch_movie_data(movie_file_name)
    if !File.file?(movie_file_name)
      @logger.debug("Downloading #{movie_file_name}...")
      movie_content = URI.open(
        'https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
samples/moviedata.zip'
      )
      movie_json = ''
      Zip::File.open_buffer(movie_content) do |zip|
        zip.each do |entry|
          movie_json = entry.get_input_stream.read
        end
      end
    else
      movie_json = File.read(movie_file_name)
    end
    movie_data = JSON.parse(movie_json)
    # The sample file lists over 4000 movies. This returns only the first 250.
    movie_data.slice(0, 250)
  rescue StandardError => e
    puts("Failure downloading movie data:\n#{e}")
    raise
  end
```

Ejecutar un escenario interactivo para crear la tabla y realizar acciones en ella.

```
  table_name = "doc-example-table-movies-#{rand(10**4)}"
  scaffold = Scaffold.new(table_name)
  dynamodb_wrapper = DynamoDBBasics.new(table_name)
```

```
 new_step(1, 'Create a new DynamoDB table if none already exists.')
 unless scaffold.exists?(table_name)
   puts("\nNo such table: #{table_name}. Creating it...")
   scaffold.create_table(table_name)
   print "Done!\n".green
 end

 new_step(2, 'Add a new record to the DynamoDB table.')
 my_movie = {}
 my_movie[:title] = CLI::UI::Prompt.ask('Enter the title of a movie to add to
the table. E.g. The Matrix')
 my_movie[:year] = CLI::UI::Prompt.ask('What year was it released? E.g.
1989').to_i
 my_movie[:rating] = CLI::UI::Prompt.ask('On a scale of 1 - 10, how do you rate
it? E.g. 7').to_i
 my_movie[:plot] = CLI::UI::Prompt.ask('Enter a brief summary of the plot. E.g.
A man awakens to a new reality.')
 dynamodb_wrapper.add_item(my_movie)
 puts("\nNew record added:")
 puts JSON.pretty_generate(my_movie).green
 print "Done!\n".green

 new_step(3, 'Update a record in the DynamoDB table.')
 my_movie[:rating] = CLI::UI::Prompt.ask("Let's update the movie you added with
a new rating, e.g. 3:").to_i
 response = dynamodb_wrapper.update_item(my_movie)
 puts("Updated '#{my_movie[:title]}' with new attributes:")
 puts JSON.pretty_generate(response).green
 print "Done!\n".green

 new_step(4, 'Get a record from the DynamoDB table.')
 puts("Searching for #{my_movie[:title]} (#{my_movie[:year]})...")
 response = dynamodb_wrapper.get_item(my_movie[:title], my_movie[:year])
 puts JSON.pretty_generate(response).green
 print "Done!\n".green

 new_step(5, 'Write a batch of items into the DynamoDB table.')
 download_file = 'moviedata.json'
 puts("Downloading movie database to #{download_file}...")
 movie_data = scaffold.fetch_movie_data(download_file)
 puts("Writing movie data from #{download_file} into your table...")
 scaffold.write_batch(movie_data)
 puts("Records added: #{movie_data.length}.")
```

```
 print "Done!\n".green

 new_step(5, 'Query for a batch of items by key.')
 loop do
   release_year = CLI::UI::Prompt.ask('Enter a year between 1972 and 2018, e.g.
1999:').to_i
   results = dynamodb_wrapper.query_items(release_year)
   if results.any?
     puts("There were #{results.length} movies released in #{release_year}:")
     results.each do |movie|
       print "\t #{movie['title']}".green
     end
     break
   else
     continue = CLI::UI::Prompt.ask("Found no movies released in
#{release_year}! Try another year? (y/n)")
     break unless continue.eql?('y')
   end
 end
 print "\nDone!\n".green

 new_step(6, 'Scan for a batch of items using a filter expression.')
 years = {}
 years[:start] = CLI::UI::Prompt.ask('Enter a starting year between 1972 and
2018:')
 years[:end] = CLI::UI::Prompt.ask('Enter an ending year between 1972 and
2018:')
 releases = dynamodb_wrapper.scan_items(years)
 if !releases.empty?
   puts("Found #{releases.length} movies.")
   count = Question.ask(
     'How many do you want to see? ', method(:is_int), in_range(1,
releases.length)
   )
   puts("Here are your #{count} movies:")
   releases.take(count).each do |release|
     puts("\t#{release['title']}")
   end
 else
   puts("I don't know about any movies released between #{years[:start]} "\
        "and #{years[:end]}.")
 end
 print "\nDone!\n".green
```

```
  new_step(7, 'Delete an item from the DynamoDB table.')
  answer = CLI::UI::Prompt.ask("Do you want to remove '#{my_movie[:title]}'? (y/
n) ")
  if answer.eql?('y')
    dynamodb_wrapper.delete_item(my_movie[:title], my_movie[:year])
    puts("Removed '#{my_movie[:title]}' from the table.")
    print "\nDone!\n".green
  end

  new_step(8, 'Delete the DynamoDB table.')
  answer = CLI::UI::Prompt.ask('Delete the table? (y/n)')
  if answer.eql?('y')
    scaffold.delete_table
    puts("Deleted #{table_name}.")
  else
    puts("Don't forget to delete the table when you're done!")
  end
  print "\nThanks for watching!\n".green
rescue Aws::Errors::ServiceError
  puts('Something went wrong with the demo.')
rescue Errno::ENOENT
  true
end
```

- Para obtener información sobre la API, consulte los siguientes temas en la referencia de la API de AWS SDK para Ruby.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Examen
  - UpdateItem

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    " Create an Amazon Dynamo DB table.

    TRY.
        DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
        DATA(lo_dyn) = /aws1/cl_dyn_factory=>create( lo_session ).
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
        DATA(oo_result) = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
 returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
```

```
        " It throws exception if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
 { lo_resourceinuseex->av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.

    " Describe table
    TRY.
        DATA(lo_table) = lo_dyn->describetable( iv_tablename = iv_table_name ).
        DATA(lv_tablename) = lo_table->get_table( )->ask_tablename( ).
        MESSAGE 'The table name is ' && lv_tablename TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table does not exist' TYPE 'E'.
    ENDTRY.

    " Put items into the table.
    TRY.
        DATA(lo_resp_putitem) = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Jaws' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1975' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '7.5' }| ) ) )
          ) ).
        lo_resp_putitem = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s = 'Star
 Wars' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1978' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
```

```
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '8.1' }| ) ) )
          ) ).
        lo_resp_putitem = lo_dyn->putitem(
          iv_tablename = iv_table_name
          it_item      = VALUE /aws1/
cl_dynattributevalue=>tt_putiteminputattributemap(
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Speed' ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '1994' }| ) ) )
            ( VALUE /aws1/cl_dynattributevalue=>ts_putiteminputattrmap_maprow(
              key = 'rating' value = NEW /aws1/cl_dynattributevalue( iv_n = |
{ '7.9' }| ) ) )
          ) ).
        " TYPE REF TO /AWSEX/CL_AWS1_dyn_PUT_ITEM_OUTPUT
        MESSAGE '3 rows inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.

    " Get item from table.
    TRY.
        DATA(lo_resp_getitem) = lo_dyn->getitem(
          iv_tablename               = iv_table_name
          it_key                     = VALUE /aws1/cl_dynattributevalue=>tt_key(
           ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
             key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 'Jaws' ) ) )
           ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
             key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n =
 '1975' ) ) )
          ) ).
        DATA(lt_attr) = lo_resp_getitem->get_item( ).
        DATA(lo_title) = lt_attr[ key = 'title' ]-value.
        DATA(lo_year) = lt_attr[ key = 'year' ]-value.
        DATA(lo_rating) = lt_attr[ key = 'year' ]-value.
```

```
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Query item from table.
    TRY.
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
             ( NEW /aws1/cl_dynattributevalue( iv_n = '1975' ) ) ).
        DATA(lt_keyconditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
            it_attributevaluelist = lt_attributelist
            iv_comparisonoperator = |EQ|
          ) ) ) ).
        DATA(lo_query_result) = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_keyconditions ).
        DATA(lt_items) = lo_query_result->get_items( ).
        READ TABLE lo_query_result->get_items( ) INTO DATA(lt_item) INDEX 1.
        lo_title = lt_item[ key = 'title' ]-value.
        lo_year = lt_item[ key = 'year' ]-value.
        lo_rating = lt_item[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Scan items from table.
    TRY.
        DATA(lo_scan_result) = lo_dyn->scan( iv_tablename = iv_table_name ).
        lt_items = lo_scan_result->get_items( ).
        " Read the first item and display the attributes.
        READ TABLE lo_query_result->get_items( ) INTO lt_item INDEX 1.
        lo_title = lt_item[ key = 'title' ]-value.
        lo_year = lt_item[ key = 'year' ]-value.
        lo_rating = lt_item[ key = 'rating' ]-value.
        MESSAGE 'Movie name is: ' && lo_title->get_s( ) TYPE 'I'.
```

```
        MESSAGE 'Movie year is: ' && lo_year->get_n( ) TYPE 'I'.
        MESSAGE 'Movie rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.

    " Update items from table.
    TRY.
        DATA(lt_attributeupdates) = VALUE /aws1/
cl_dynattrvalueupdate=>tt_attributeupdates(
          ( VALUE /aws1/cl_dynattrvalueupdate=>ts_attributeupdates_maprow(
          key = 'rating' value = NEW /aws1/cl_dynattrvalueupdate(
            io_value  = NEW /aws1/cl_dynattributevalue( iv_n = '7.6' )
            iv_action = |PUT| ) ) ) ).
        DATA(lt_key) = VALUE /aws1/cl_dynattributevalue=>tt_key(
          ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
            key = 'year' value = NEW /aws1/cl_dynattributevalue( iv_n =
 '1975' ) ) )
          ( VALUE /aws1/cl_dynattributevalue=>ts_key_maprow(
            key = 'title' value = NEW /aws1/cl_dynattributevalue( iv_s =
 '1980' ) ) ) ).
        DATA(lo_resp) = lo_dyn->updateitem(
          iv_tablename        = iv_table_name
          it_key              = lt_key
          it_attributeupdates = lt_attributeupdates ).
        MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
 TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.

    " Delete table.
    TRY.
        lo_dyn->deletetable( iv_tablename = iv_table_name ).
        lo_dyn->get_waiter( )->tablenotexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table deleted.' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
```

```
      CATCH /aws1/cx_dynresourceinuseex.
        MESSAGE 'The table cannot be deleted as it is in use' TYPE 'E'.
    ENDTRY.
```

- Para detalles acerca de la API, consulte los siguientes temas en la Referencia de la API del SDK de AWS para SAP ABAP.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Examen
  - UpdateItem

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Una clase de Swift que gestiona las llamadas de DynamoDB al SDK para Swift.

```
import AWSDynamoDB
import Foundation

/// An enumeration of error codes representing issues that can arise when using
/// the `MovieTable` class.
enum MoviesError: Error {
    /// The specified table wasn't found or couldn't be created.
```

```
    case TableNotFound
    /// The specified item wasn't found or couldn't be created.
    case ItemNotFound
    /// The Amazon DynamoDB client is not properly initialized.
    case UninitializedClient
    /// The table status reported by Amazon DynamoDB is not recognized.
    case StatusUnknown
    /// One or more specified attribute values are invalid or missing.
    case InvalidAttributes
}

/// A class representing an Amazon DynamoDB table containing movie
/// information.
public class MovieTable {
    var ddbClient: DynamoDBClient?
    let tableName: String

    /// Create an object representing a movie table in an Amazon DynamoDB
    /// database.
    ///
    /// - Parameters:
    ///   - region: The optional Amazon Region to create the database in.
    ///   - tableName: The name to assign to the table. If not specified, a
    ///     random table name is generated automatically.
    ///
    /// > Note: The table is not necessarily available when this function
    /// returns. Use `tableExists()` to check for its availability, or
    /// `awaitTableActive()` to wait until the table's status is reported as
    /// ready to use by Amazon DynamoDB.
    ///
    init(region: String? = nil, tableName: String) async throws {
        do {
            let config = try await DynamoDBClient.DynamoDBClientConfiguration()
            if let region = region {
                config.region = region
            }

            self.ddbClient = DynamoDBClient(config: config)
            self.tableName = tableName

            try await self.createTable()
        } catch {
            print("ERROR: ", dump(error, name: "Initializing Amazon
 DynamoDBClient client"))
```

```
            throw error
        }
    }

    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = CreateTableInput(
                attributeDefinitions: [
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"year", attributeType: .n),
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
"title", attributeType: .s)
                ],
                billingMode: DynamoDBClientTypes.BillingMode.payPerRequest,
                keySchema: [
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
keyType: .hash),
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
keyType: .range)
                ],
                tableName: self.tableName
            )
            let output = try await client.createTable(input: input)
            if output.tableDescription == nil {
                throw MoviesError.TableNotFound
            }
        } catch {
            print("ERROR: createTable:", dump(error))
            throw error
        }
    }


    /// Check to see if the table exists online yet.
    ///
    /// - Returns: `true` if the table exists, or `false` if not.
    ///
```

```
    func tableExists() async throws -> Bool {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DescribeTableInput(
                tableName: tableName
            )
            let output = try await client.describeTable(input: input)
            guard let description = output.table else {
                throw MoviesError.TableNotFound
            }

            return description.tableName == self.tableName
        } catch {
            print("ERROR: tableExists:", dump(error))
            throw error
        }
    }


    ///
    /// Waits for the table to exist and for its status to be active.
    ///
    func awaitTableActive() async throws {
        while try (await self.tableExists() == false) {
            do {
                let duration = UInt64(0.25 * 1_000_000_000) // Convert .25
seconds to nanoseconds.
                try await Task.sleep(nanoseconds: duration)
            } catch {
                print("Sleep error:", dump(error))
            }
        }

        while try (await self.getTableStatus() != .active) {
            do {
                let duration = UInt64(0.25 * 1_000_000_000) // Convert .25
seconds to nanoseconds.
                try await Task.sleep(nanoseconds: duration)
            } catch {
                print("Sleep error:", dump(error))
            }
```

```
        }
    }


    ///
    /// Deletes the table from Amazon DynamoDB.
    ///
    func deleteTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteTableInput(
                tableName: self.tableName
            )
            _ = try await client.deleteTable(input: input)
        } catch {
            print("ERROR: deleteTable:", dump(error))
            throw error
        }
    }


    /// Get the table's status.
    ///
    /// - Returns: The table status, as defined by the
    ///   `DynamoDBClientTypes.TableStatus` enum.
    ///
    func getTableStatus() async throws -> DynamoDBClientTypes.TableStatus {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DescribeTableInput(
                tableName: self.tableName
            )
            let output = try await client.describeTable(input: input)
            guard let description = output.table else {
                throw MoviesError.TableNotFound
            }
            guard let status = description.tableStatus else {
                throw MoviesError.StatusUnknown
```

```
            }
            return status
        } catch {
            print("ERROR: getTableStatus:", dump(error))
            throw error
        }
    }


    /// Populate the movie database from the specified JSON file.
    ///
    /// - Parameter jsonPath: Path to a JSON file containing movie data.
    ///
    func populate(jsonPath: String) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Create a Swift `URL` and use it to load the file into a `Data`
            // object. Then decode the JSON into an array of `Movie` objects.

            let fileUrl = URL(fileURLWithPath: jsonPath)
            let jsonData = try Data(contentsOf: fileUrl)

            var movieList = try JSONDecoder().decode([Movie].self, from:
jsonData)

            // Truncate the list to the first 200 entries or so for this example.

            if movieList.count > 200 {
                movieList = Array(movieList[...199])
            }

            // Before sending records to the database, break the movie list into
            // 25-entry chunks, which is the maximum size of a batch item
request.

            let count = movieList.count
            let chunks = stride(from: 0, to: count, by: 25).map {
                Array(movieList[$0 ..< Swift.min($0 + 25, count)])
            }
```

```
            // For each chunk, create a list of write request records and
populate
            // them with `PutRequest` requests, each specifying one movie from
the
            // chunk. Once the chunk's items are all in the `PutRequest` list,
            // send them to Amazon DynamoDB using the
            // `DynamoDBClient.batchWriteItem()` function.

            for chunk in chunks {
                var requestList: [DynamoDBClientTypes.WriteRequest] = []

                for movie in chunk {
                    let item = try await movie.getAsItem()
                    let request = DynamoDBClientTypes.WriteRequest(
                        putRequest: .init(
                            item: item
                        )
                    )
                    requestList.append(request)
                }

                let input = BatchWriteItemInput(requestItems: [tableName:
requestList])
                _ = try await client.batchWriteItem(input: input)
            }
        } catch {
            print("ERROR: populate:", dump(error))
            throw error
        }
    }


    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    ///
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Get a DynamoDB item containing the movie data.
```

```
            let item = try await movie.getAsItem()

            // Send the `PutItem` request to Amazon DynamoDB.

            let input = PutItemInput(
                item: item,
                tableName: self.tableName
            )
            _ = try await client.putItem(input: input)
        } catch {
            print("ERROR: add movie:", dump(error))
            throw error
        }
    }


    /// Given a movie's details, add a movie to the Amazon DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title as a `String`.
    ///   - year: The release year of the movie (`Int`).
    ///   - rating: The movie's rating if available (`Double`; default is
    ///     `nil`).
    ///   - plot: A summary of the movie's plot (`String`; default is `nil`,
    ///     indicating no plot summary is available).
    ///
    func add(title: String, year: Int, rating: Double? = nil,
             plot: String? = nil) async throws
    {
        do {
            let movie = Movie(title: title, year: year, rating: rating, plot:
plot)
            try await self.add(movie: movie)
        } catch {
            print("ERROR: add with fields:", dump(error))
            throw error
        }
    }


    /// Return a `Movie` record describing the specified movie from the Amazon
    /// DynamoDB table.
    ///
    /// - Parameters:
```

```
    ///   - title: The movie's title (`String`).
    ///   - year: The movie's release year (`Int`).
    ///
    /// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
    ///
    /// - Returns: A `Movie` record with the movie's details.
    func get(title: String, year: Int) async throws -> Movie {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = GetItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            let output = try await client.getItem(input: input)
            guard let item = output.item else {
                throw MoviesError.ItemNotFound
            }

            let movie = try Movie(withItem: item)
            return movie
        } catch {
            print("ERROR: get:", dump(error))
            throw error
        }
    }


    /// Get all the movies released in the specified year.
    ///
    /// - Parameter year: The release year of the movies to return.
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }
```

```
            let input = QueryInput(
                expressionAttributeNames: [
                    "#y": "year"
                ],
                expressionAttributeValues: [
                    ":y": .n(String(year))
                ],
                keyConditionExpression: "#y = :y",
                tableName: self.tableName
            )
            // Use "Paginated" to get all the movies.
            // This lets the SDK handle the 'lastEvaluatedKey' property in
"QueryOutput".

            let pages = client.queryPaginated(input: input)

            var movieList: [Movie] = []
            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Convert the found movies into `Movie` objects and return an
array
                // of them.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList
        } catch {
            print("ERROR: getMovies:", dump(error))
            throw error
        }
    }


    /// Return an array of `Movie` objects released in the specified range of
    /// years.
    ///
```

```
    /// - Parameters:
    ///   - firstYear: The first year of movies to return.
    ///   - lastYear: The last year of movies to return.
    ///   - startKey: A starting point to resume processing; always use `nil`.
    ///
    /// - Returns: An array of `Movie` objects describing the matching movies.
    ///
    /// > Note: The `startKey` parameter is used by this function when
    ///   recursively calling itself, and should always be `nil` when calling
    ///   directly.
    ///
    func getMovies(firstYear: Int, lastYear: Int,
                   startKey: [Swift.String: DynamoDBClientTypes.AttributeValue]?
= nil)
        async throws -> [Movie]
    {
        do {
            var movieList: [Movie] = []

            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = ScanInput(
                consistentRead: true,
                exclusiveStartKey: startKey,
                expressionAttributeNames: [
                    "#y": "year" // `year` is a reserved word, so use `#y`
instead.
                ],
                expressionAttributeValues: [
                    ":y1": .n(String(firstYear)),
                    ":y2": .n(String(lastYear))
                ],
                filterExpression: "#y BETWEEN :y1 AND :y2",
                tableName: self.tableName
            )

            let pages = client.scanPaginated(input: input)

            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
```

```
                }

                // Build an array of `Movie` objects for the returned items.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
                }
            }
            return movieList

        } catch {
            print("ERROR: getMovies with scan:", dump(error))
            throw error
        }
    }


    /// Update the specified movie with new `rating` and `plot` information.
    ///
    /// - Parameters:
    ///   - title: The title of the movie to update.
    ///   - year: The release year of the movie to update.
    ///   - rating: The new rating for the movie.
    ///   - plot: The new plot summary string for the movie.
    ///
    /// - Returns: An array of mappings of attribute names to their new
    ///   listing each item actually changed. Items that didn't need to change
    ///   aren't included in this list. `nil` if no changes were made.
    ///
    func update(title: String, year: Int, rating: Double? = nil, plot: String? =
nil) async throws
        -> [Swift.String: DynamoDBClientTypes.AttributeValue]?
    {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Build the update expression and the list of expression attribute
            // values. Include only the information that's changed.

            var expressionParts: [String] = []
```

```
            var attrValues: [Swift.String: DynamoDBClientTypes.AttributeValue] =
[:]

            if rating != nil {
                expressionParts.append("info.rating=:r")
                attrValues[":r"] = .n(String(rating!))
            }
            if plot != nil {
                expressionParts.append("info.plot=:p")
                attrValues[":p"] = .s(plot!)
            }
            let expression = "set \(expressionParts.joined(separator: ", "))"

            let input = UpdateItemInput(
                // Create substitution tokens for the attribute values, to ensure
                // no conflicts in expression syntax.
                expressionAttributeValues: attrValues,
                // The key identifying the movie to update consists of the
release
                // year and title.
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                returnValues: .updatedNew,
                tableName: self.tableName,
                updateExpression: expression
            )
            let output = try await client.updateItem(input: input)

            guard let attributes: [Swift.String:
DynamoDBClientTypes.AttributeValue] = output.attributes else {
                throw MoviesError.InvalidAttributes
            }
            return attributes
        } catch {
            print("ERROR: update:", dump(error))
            throw error
        }
    }


    /// Delete a movie, given its title and release year.
    ///
```

```
    /// - Parameters:
    ///   - title: The movie's title.
    ///   - year: The movie's release year.
    ///
    func delete(title: String, year: Int) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            _ = try await client.deleteItem(input: input)
        } catch {
            print("ERROR: delete:", dump(error))
            throw error
        }
    }
}
```

Las estructuras que utiliza la clase MovieTable para representar películas.

```
import Foundation
import AWSDynamoDB

/// The optional details about a movie.
public struct Details: Codable {
    /// The movie's rating, if available.
    var rating: Double?
    /// The movie's plot, if available.
    var plot: String?
}

/// A structure describing a movie. The `year` and `title` properties are
/// required and are used as the key for Amazon DynamoDB operations. The
/// `info` sub-structure's two properties, `rating` and `plot`, are optional.
```

```
public struct Movie: Codable {
    /// The year in which the movie was released.
    var year: Int
    /// The movie's title.
    var title: String
    /// A `Details` object providing the optional movie rating and plot
    /// information.
    var info: Details

    /// Create a `Movie` object representing a movie, given the movie's
    /// details.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The year in which the movie was released (`Int`).
    ///   - rating: The movie's rating (optional `Double`).
    ///   - plot: The movie's plot (optional `String`)
    init(title: String, year: Int, rating: Double? = nil, plot: String? = nil) {
        self.title = title
        self.year = year

        self.info = Details(rating: rating, plot: plot)
    }

    /// Create a `Movie` object representing a movie, given the movie's
    /// details.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The year in which the movie was released (`Int`).
    ///   - info: The optional rating and plot information for the movie in a
    ///     `Details` object.
    init(title: String, year: Int, info: Details?){
        self.title = title
        self.year = year

        if info != nil {
            self.info = info!
        } else {
            self.info = Details(rating: nil, plot: nil)
        }
    }

    ///
```

```
    /// Return a new `MovieTable` object, given an array mapping string to Amazon
    /// DynamoDB attribute values.
    ///
    /// - Parameter item: The item information provided to the form used by
    ///   DynamoDB. This is an array of strings mapped to
    ///   `DynamoDBClientTypes.AttributeValue` values.
    init(withItem item: [Swift.String:DynamoDBClientTypes.AttributeValue]) throws
{
        // Read the attributes.

        guard let titleAttr = item["title"],
              let yearAttr = item["year"] else {
            throw MoviesError.ItemNotFound
        }
        let infoAttr = item["info"] ?? nil

        // Extract the values of the title and year attributes.

        if case .s(let titleVal) = titleAttr {
            self.title = titleVal
        } else {
            throw MoviesError.InvalidAttributes
        }

        if case .n(let yearVal) = yearAttr {
            self.year = Int(yearVal)!
        } else {
            throw MoviesError.InvalidAttributes
        }

        // Extract the rating and/or plot from the `info` attribute, if
        // they're present.

        var rating: Double? = nil
        var plot: String? = nil

        if infoAttr != nil, case .m(let infoVal) = infoAttr {
            let ratingAttr = infoVal["rating"] ?? nil
            let plotAttr = infoVal["plot"] ?? nil

            if ratingAttr != nil, case .n(let ratingVal) = ratingAttr {
                rating = Double(ratingVal) ?? nil
            }
            if plotAttr != nil, case .s(let plotVal) = plotAttr {
```

```
                plot = plotVal
            }
        }

        self.info = Details(rating: rating, plot: plot)
    }

    ///
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
[Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
            }
        }
        item["info"] = .m(details)

        return item
    }
}
```

Un programa que utiliza la clase MovieTable para acceder a una base de datos de DynamoDB.

```
import ArgumentParser
import ClientRuntime
import Foundation

import AWSDynamoDB


@testable import MovieList

extension String {
    // Get the directory if the string is a file path.
    func directory() -> String {
        guard let lastIndex = lastIndex(of: "/") else {
            print("Error: String directory separator not found.")
            return ""
        }
        return String(self[...lastIndex])
    }
}

struct ExampleCommand: ParsableCommand {
    @Argument(help: "The path of the sample movie data JSON file.")
    var jsonPath: String = #file.directory() + "../../../../../resources/
sample_files/movies.json"

    @Option(help: "The AWS Region to run AWS API calls in.")
    var awsRegion: String?

    @Option(
        help: ArgumentHelp("The level of logging for the Swift SDK to perform."),
        completion: .list([
            "critical",
            "debug",
            "error",
            "info",
            "notice",
            "trace",
            "warning"
        ])
    )
```

```
    var logLevel: String = "error"

    /// Configuration details for the command.
    static var configuration = CommandConfiguration(
        commandName: "basics",
        abstract: "A basic scenario demonstrating the usage of Amazon DynamoDB.",
        discussion: """
        An example showing how to use Amazon DynamoDB to perform a series of
        common database activities on a simple movie database.
        """
    )

    /// Called by ``main()`` to asynchronously run the AWS example.
    func runAsync() async throws {
        print("Welcome to the AWS SDK for Swift basic scenario for Amazon
DynamoDB!")

        //========================================================================
        // 1. Create the table. The Amazon DynamoDB table is represented by
        //    the `MovieTable` class.
        //========================================================================

        let tableName = "ddb-movies-sample-\(Int.random(in: 1 ... Int.max))"

        print("Creating table \"\(tableName)\"...")

        let movieDatabase = try await MovieTable(region: awsRegion,
                                                 tableName: tableName)

        print("\nWaiting for table to be ready to use...")
        try await movieDatabase.awaitTableActive()

        //========================================================================
        // 2. Add a movie to the table.
        //========================================================================

        print("\nAdding a movie...")
        try await movieDatabase.add(title: "Avatar: The Way of Water", year:
2022)
        try await movieDatabase.add(title: "Not a Real Movie", year: 2023)

        //========================================================================
        // 3. Update the plot and rating of the movie using an update
        //    expression.
```

```
        //============================================================================

        print("\nAdding details to the added movie...")
        _ = try await movieDatabase.update(title: "Avatar: The Way of Water",
year: 2022,
                                           rating: 9.2, plot: "It's a sequel.")

        //============================================================================
        // 4. Populate the table from the JSON file.
        //============================================================================

        print("\nPopulating the movie database from JSON...")
        try await movieDatabase.populate(jsonPath: jsonPath)

        //============================================================================
        // 5. Get a specific movie by key. In this example, the key is a
        //    combination of `title` and `year`.
        //============================================================================

        print("\nLooking for a movie in the table...")
        let gotMovie = try await movieDatabase.get(title: "This Is the End",
year: 2013)

        print("Found the movie \"\(gotMovie.title)\", released in
\(gotMovie.year).")
        print("Rating: \(gotMovie.info.rating ?? 0.0).")
        print("Plot summary: \(gotMovie.info.plot ?? "None.")")

        //============================================================================
        // 6. Delete a movie.
        //============================================================================

        print("\nDeleting the added movie...")
        try await movieDatabase.delete(title: "Avatar: The Way of Water", year:
2022)

        //============================================================================
        // 7. Use a query with a key condition expression to return all movies
        //    released in a given year.
        //============================================================================

        print("\nGetting movies released in 1994...")
        let movieList = try await movieDatabase.getMovies(fromYear: 1994)
        for movie in movieList {
```

```
            print("    \(movie.title)")
        }

        //===========================================================================
        // 8. Use `scan()` to return movies released in a range of years.
        //===========================================================================

        print("\nGetting movies released between 1993 and 1997...")
        let scannedMovies = try await movieDatabase.getMovies(firstYear: 1993,
 lastYear: 1997)
        for movie in scannedMovies {
            print("    \(movie.title) (\(movie.year))")
        }

        //===========================================================================
        // 9. Delete the table.
        //===========================================================================

        print("\nDeleting the table...")
        try await movieDatabase.deleteTable()
    }
}

@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Swift.
  - BatchWriteItem

- CreateTable
- DeleteItem
- DeleteTable
- DescribeTable
- GetItem
- PutItem
- Query
- Examen
- UpdateItem

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Acciones de DynamoDB con los SDK de AWS

En los siguientes ejemplos de código se demuestra cómo realizar acciones individuales de DynamoDB con los SDK de AWS. En cada ejemplo se incluye un enlace a GitHub, con instrucciones de configuración y ejecución del código.

Estos fragmentos llaman a la API de DynamoDB y son fragmentos de código de programas más grandes que se deben ejecutar en contexto. Puede ver las acciones en contexto en Escenarios para DynamoDB con los SDK de AWS.

Los siguientes ejemplos incluyen solo las acciones que se utilizan con mayor frecuencia. Para ver una lista completa, consulte la referencia de la API de Amazon DynamoDB.

Ejemplos

- Uso de BatchExecuteStatement con un AWS SDK
- Uso de BatchGetItem con un SDK de AWS o la CLI
- Uso de BatchWriteItem con un SDK de AWS o la CLI
- Uso de CreateTable con un SDK de AWS o la CLI
- Uso de DeleteItem con un SDK de AWS o la CLI
- Uso de DeleteTable con un SDK de AWS o la CLI

- Uso de DescribeTable con un SDK de AWS o la CLI
- Uso de DescribeTimeToLive con un SDK de AWS o la CLI
- Uso de ExecuteStatement con un SDK de AWS
- Uso de GetItem con un SDK de AWS o la CLI
- Uso de ListTables con un SDK de AWS o la CLI
- Uso de PutItem con un SDK de AWS o la CLI
- Uso de Query con un SDK de AWS o la CLI
- Uso de Scan con un SDK de AWS o la CLI
- Uso de UpdateItem con un SDK de AWS o la CLI
- Uso de UpdateTable con un SDK de AWS o la CLI
- Uso de UpdateTimeToLive con un SDK de AWS o la CLI

Uso de **`BatchExecuteStatement`** con un AWS SDK

Los siguientes ejemplos de código muestran cómo utilizar `BatchExecuteStatement`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Eliminación de datos mediante DELETE de PartiQL
- Inserción de datos mediante INSERT de PartiQL
- Consultar una tabla mediante lotes de instrucciones PartiQL
- Consulta de datos mediante SELECT de PartiQL
- Actualización de datos mediante UPDATE de PartiQL

.NET

SDK para .NET

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Utilizar lotes de instrucciones INSERT para agregar elementos.

```
        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table into which the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
```

```
                                },
                            });
                        }

                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
```

```
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }
```

Utilizar lotes de instrucciones SELECT para obtener elementos.

```
        /// <summary>
        /// Gets movies from the movie table by
        /// using an Amazon DynamoDB PartiQL SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year1">The year of the first movie.</param>
        /// <param name="year2">The year of the second movie.</param>
        /// <returns>True if successful.</returns>
        public static async Task<bool> GetBatch(
            string tableName,
            string title1,
            string title2,
            int year1,
            int year2)
        {
            var getBatch = $"SELECT * FROM {tableName} WHERE title = ? AND year
 = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
```

```
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            if (response.Responses.Count > 0)
            {
                response.Responses.ForEach(r =>
                {
                    if (r.Item.Any())
                    {

Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
                    }
                });
                return true;
            }
            else
            {
                Console.WriteLine($"Couldn't find either {title1} or {title2}.");
                return false;
            }

        }
```

Utilizar lotes de instrucciones UPDATE para actualizar elementos.

```
        /// <summary>
        /// Updates information for multiple movies.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// movies to be updated.</param>
        /// <param name="producer1">The producer name for the first movie
        /// to update.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year that the first movie was released.</
param>
        /// <param name="producer2">The producer name for the second
        /// movie to update.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year that the second movie was released.</
param>
        /// <returns>A Boolean value that indicates the success of the update.</
returns>
        public static async Task<bool> UpdateBatch(
            string tableName,
            string producer1,
            string title1,
            int year1,
            string producer2,
            string title2,
            int year2)
        {

            string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title
 = ? AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer1 },
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },
```

```
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer2 },
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Utilizar lotes de instrucciones DELETE para eliminar elementos.

```
        /// <summary>
        /// Deletes multiple movies using a PartiQL BatchExecuteAsync
        /// statement.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// moves that will be deleted.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year the first movie was released.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year the second movie was released.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeleteBatch(
            string tableName,
            string title1,
            int year1,
            string title2,
            int year2)
        {
```

```
            string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND
year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Para obtener información sobre la API, consulte BatchExecuteStatement en la referencia de la API de AWS SDK para .NET.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Utilizar lotes de instrucciones INSERT para agregar elementos.

```
    // 2. Add multiple movies using "Insert" statements. (BatchExecuteStatement)
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::vector<Aws::String> titles;
    std::vector<float> ratings;
    std::vector<int> years;
    std::vector<Aws::String> plots;
    Aws::String doAgain = "n";
    do {
        Aws::String aTitle = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        titles.push_back(aTitle);
        int aYear = askQuestionForInt("What year was it released? ");
        years.push_back(aYear);
        float aRating = askQuestionForFloatRange(
                "On a scale of 1 - 10, how do you rate it? ",
                1, 10);
        ratings.push_back(aRating);
        Aws::String aPlot = askQuestion("Summarize the plot for me: ");
        plots.push_back(aPlot);

        doAgain = askQuestion(Aws::String("Would you like to add more movies? (y/
n) "));
    } while (doAgain == "y");

    std::cout << "Adding " << titles.size()
              << (titles.size() == 1 ? " movie " : " movies ")
              << "to the table using a batch \"INSERT\" statement." << std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
```

```
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);

            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));

            // Create attribute for the info map.
            Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute
= Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            ratingAttribute->SetN(ratings[i]);
            infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            plotAttribute->SetS(plots[i]);
            infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
            attributes.push_back(infoMapAttribute);
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
```

```
            std::cerr << "Failed to add the movies: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }
```

Utilizar lotes de instrucciones SELECT para obtener elementos.

```
    // 3. Get the data for multiple movies using "Select" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (outcome.IsSuccess()) {
            const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();
```

```
            const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

            for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

                printMovieInfo(item);
            }
        }
        else {
            std::cerr << "Failed to retrieve the movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

Utilizar lotes de instrucciones UPDATE para actualizar elementos.

```
    // 4. Update the data for multiple movies using "Update" statements.
(BatchExecuteStatement)

    for (size_t i = 0; i < titles.size(); ++i) {
        ratings[i] = askQuestionForFloatRange(
                Aws::String("\nLet's update your the movie, \"") + titles[i] +
                ".\nYou rated it  " + std::to_string(ratings[i])
                + ", what new rating would you give it? ", 1, 10);
    }

    std::cout << "Updating the movie with a batch \"UPDATE\" statement." <<
std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";
```

```
        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);

            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetN(ratings[i]));
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);
        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

Utilizar lotes de instrucciones DELETE para eliminar elementos.

```
    // 6. Delete multiple movies using "Delete" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());
```

```
        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movies: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

- Para obtener información sobre la API, consulte BatchExecuteStatement en la referencia de la API de AWS SDK para C++.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Defina una estructura de receptor de funciones para el ejemplo.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

Utilizar lotes de instrucciones INSERT para agregar elementos.

```
// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies
 to the
// DynamoDB table.
func (runner PartiQLRunner) AddMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year, movie.Info})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(fmt.Sprintf(
    "INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
 runner.TableName)),
   Parameters: params,
  }
```

```
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't insert a batch of items with PartiQL. Here's why: %v\n",
 err)
 }
 return err
}
```

Utilizar lotes de instrucciones SELECT para obtener elementos.

```
// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
 from
// the DynamoDB table by title and year.
func (runner PartiQLRunner) GetMovieBatch(ctx context.Context, movies []Movie)
 ([]Movie, error) {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 output, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 var outMovies []Movie
```

```
 if err != nil {
  log.Printf("Couldn't get a batch of items with PartiQL. Here's why: %v\n", err)
 } else {
  for _, response := range output.Responses {
   var movie Movie
   err = attributevalue.UnmarshalMap(response.Item, &movie)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    outMovies = append(outMovies, movie)
   }
  }
 }
 return outMovies, err
}
```

Utilizar lotes de instrucciones UPDATE para actualizar elementos.

```
// UpdateMovieBatch runs a batch of PartiQL UPDATE statements to update the
 rating of
// multiple movies that already exist in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovieBatch(ctx context.Context, movies []Movie,
 ratings []float64) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{ratings[index],
 movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
```

```
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

Utilizar lotes de instrucciones DELETE para eliminar elementos.

```
// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
 movies
// from the DynamoDB table.
func (runner PartiQLRunner) DeleteMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

Defina una estructura Película para utilizar en este ejemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}
```

```
// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obtener información sobre la API, consulte BatchExecuteStatement en la referencia de la API de AWS SDK para Go.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear un lote de elementos con PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const breakfastFoods = ["Eggs", "Bacon", "Sausage"];
  const command = new BatchExecuteStatementCommand({
    Statements: breakfastFoods.map((food) => ({
      Statement: `INSERT INTO BreakfastFoods value {'Name':?}`,
      Parameters: [food],
    })),
```

```
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Obtener un lote de elementos con PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Teaspoons"],
        ConsistentRead: true,
      },
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Grams"],
        ConsistentRead: true,
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Actualizar un lote de elementos con PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const eggUpdates = [
    ["duck", "fried"],
    ["chicken", "omelette"],
  ];
  const command = new BatchExecuteStatementCommand({
    Statements: eggUpdates.map((change) => ({
      Statement: "UPDATE Eggs SET Style=? where Variety=?",
      Parameters: [change[1], change[0]],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Eliminar un lote de elementos con PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
```

```
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Grape"],
      },
      {
        Statement: "DELETE FROM Flavors where Name=?",
        Parameters: ["Strawberry"],
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obtener información sobre la API, consulte BatchExecuteStatement en la referencia de la API de AWS SDK para JavaScript.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    public function getItemByPartiQLBatch(string $tableName, array $keys): Result
    {
        $statements = [];
        foreach ($keys as $key) {
            list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);
            $statements[] = [
                'Statement' => "$statement",
                'Parameters' => $parameters,
            ];
        }
```

```
        return $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => $statements,
        ]);
    }

    public function insertItemByPartiQLBatch(string $statement, array
$parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function updateItemByPartiQLBatch(string $statement, array
$parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function deleteItemByPartiQLBatch(string $statement, array
$parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }
```

- Para obtener información sobre la API, consulte BatchExecuteStatement en la referencia de la API de AWS SDK para PHP.

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class PartiQLBatchWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statements, param_list):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statements: The batch of PartiQL statements.
        :param param_list: The batch of PartiQL parameters that are associated
 with
```

```
                            each statement. This list must be in the same order as
the
                            statements.
        :return: The responses returned from running the statements, if any.
        """
        try:
            output = self.dyn_resource.meta.client.batch_execute_statement(
                Statements=[
                    {"Statement": statement, "Parameters": params}
                    for statement, params in zip(statements, param_list)
                ]
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute batch of PartiQL statements because the
table "
                    "does not exist."
                )
            else:
                logger.error(
                    "Couldn't execute batch of PartiQL statements. Here's why:
%s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

- Para obtener información sobre la API, consulte BatchWriteItem en Referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Leer un lote de elementos con PartiQL.

```
class DynamoDBPartiQLBatch
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Selects a batch of items from a table using PartiQL
  #
  # @param batch_titles [Array] Collection of movie titles
  # @return [Aws::DynamoDB::Types::BatchExecuteStatementOutput]
  def batch_execute_select(batch_titles)
    request_items = batch_titles.map do |title, year|
      {
        statement: "SELECT * FROM \"#{@table.name}\" WHERE title=? and year=?",
        parameters: [title, year]
      }
    end
    @dynamodb.client.batch_execute_statement({ statements: request_items })
  end
```

Eliminar un lote de elementos con PartiQL.

```
class DynamoDBPartiQLBatch
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
```

```
  client = Aws::DynamoDB::Client.new(region: 'us-east-1')
  @dynamodb = Aws::DynamoDB::Resource.new(client: client)
  @table = @dynamodb.table(table_name)
end

# Deletes a batch of items from a table using PartiQL
#
# @param batch_titles [Array] Collection of movie titles
# @return [Aws::DynamoDB::Types::BatchExecuteStatementOutput]
def batch_execute_write(batch_titles)
  request_items = batch_titles.map do |title, year|
    {
      statement: "DELETE FROM \"#{@table.name}\" WHERE title=? and year=?",
      parameters: [title, year]
    }
  end
  @dynamodb.client.batch_execute_statement({ statements: request_items })
end
```

- Para obtener información sobre la API, consulte BatchExecuteStatement en la referencia de la API de AWS SDK para Ruby.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de **`BatchGetItem`** con un SDK de AWS o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `BatchGetItem`.

.NET

SDK para .NET

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;

namespace LowLevelBatchGet
{
    public class LowLevelBatchGet
    {
        private static readonly string _table1Name = "Forum";
        private static readonly string _table2Name = "Thread";

        public static async void
 RetrieveMultipleItemsBatchGet(AmazonDynamoDBClient client)
        {
            var request = new BatchGetItemRequest
            {
                RequestItems = new Dictionary<string, KeysAndAttributes>()
            {
                { _table1Name,
                  new KeysAndAttributes
                  {
                      Keys = new List<Dictionary<string, AttributeValue> >()
                      {
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                            S = "Amazon DynamoDB"
                        } }
                          },
                          new Dictionary<string, AttributeValue>()
                          {
                              { "Name", new AttributeValue {
                            S = "Amazon S3"
                        } }
                          }
                      }
                  }},
                {
                    _table2Name,
                    new KeysAndAttributes
                    {
                        Keys = new List<Dictionary<string, AttributeValue> >()
```

```
                        {
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon DynamoDB"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "DynamoDB Thread 1"
                                  } }
                            },
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon DynamoDB"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "DynamoDB Thread 2"
                                  } }
                            },
                            new Dictionary<string, AttributeValue>()
                            {
                                { "ForumName", new AttributeValue {
                                      S = "Amazon S3"
                                  } },
                                { "Subject", new AttributeValue {
                                      S = "S3 Thread 1"
                                  } }
                            }
                        }
                    }
                }
            }
            };

            BatchGetItemResponse response;
            do
            {
                Console.WriteLine("Making request");
                response = await client.BatchGetItemAsync(request);

                // Check the response.
                var responses = response.Responses; // Attribute list in the
response.
```

```
                foreach (var tableResponse in responses)
                {
                    var tableResults = tableResponse.Value;
                    Console.WriteLine("Items retrieved from table {0}",
tableResponse.Key);
                    foreach (var item1 in tableResults)
                    {
                        PrintItem(item1);
                    }
                }

                // Any unprocessed keys? could happen if you exceed
ProvisionedThroughput or some other error.
                Dictionary<string, KeysAndAttributes> unprocessedKeys =
response.UnprocessedKeys;
                foreach (var unprocessedTableKeys in unprocessedKeys)
                {
                    // Print table name.
                    Console.WriteLine(unprocessedTableKeys.Key);
                    // Print unprocessed primary keys.
                    foreach (var key in unprocessedTableKeys.Value.Keys)
                    {
                        PrintItem(key);
                    }
                }

                request.RequestItems = unprocessedKeys;
            } while (response.UnprocessedKeys.Count > 0);
        }

        private static void PrintItem(Dictionary<string, AttributeValue>
attributeList)
        {
            foreach (KeyValuePair<string, AttributeValue> kvp in attributeList)
            {
                string attributeName = kvp.Key;
                AttributeValue value = kvp.Value;

                Console.WriteLine(
                    attributeName + " " +
                    (value.S == null ? "" : "S=[" + value.S + "]") +
                    (value.N == null ? "" : "N=[" + value.N + "]") +
                    (value.SS == null ? "" : "SS=[" + string.Join(",",
value.SS.ToArray()) + "]") +
```

```
                    (value.NS == null ? "" : "NS=[" + string.Join(",",
 value.NS.ToArray()) + "]")
                    );
            }

 Console.WriteLine("*********************************************");
        }

        static void Main()
        {
            var client = new AmazonDynamoDBClient();

            RetrieveMultipleItemsBatchGet(client);
        }
    }
}
```

- Para obtener información de la API, consulte BatchGetItem en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_batch_get_item
#
# This function gets a batch of items from a DynamoDB table.
#
# Parameters:
#       -i item  -- Path to json file containing the keys of the items to get.
#
#  Returns:
#       The items as json output.
```

```
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_batch_get_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_batch_get_item"
    echo "Get a batch of items from a DynamoDB table."
    echo " -i item  -- Path to json file containing the keys of the items to
 get."
    echo ""
  }

  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
    return 1
  fi

  response=$(aws dynamodb batch-get-item \
    --request-items file://"$item")
  local error_code=${?}
```

```
  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-get-item operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
```

```
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información de la API, consulte BatchGetItem en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Batch get items from different Amazon DynamoDB tables.
/*!
  \sa batchGetItem()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::batchGetItem(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
```

```
    Aws::DynamoDB::Model::BatchGetItemRequest request;

    // Table1: Forum.
    Aws::String table1Name = "Forum";
    Aws::DynamoDB::Model::KeysAndAttributes table1KeysAndAttributes;

    // Table1: Projection expression.
    table1KeysAndAttributes.SetProjectionExpression("#n, Category, Messages,
#v");

    // Table1: Expression attribute names.
    Aws::Http::HeaderValueCollection headerValueCollection;
    headerValueCollection.emplace("#n", "Name");
    headerValueCollection.emplace("#v", "Views");
    table1KeysAndAttributes.SetExpressionAttributeNames(headerValueCollection);

    // Table1: Set key name, type, and value to search.
    std::vector<Aws::String> nameValues = {"Amazon DynamoDB", "Amazon S3"};
    for (const Aws::String &name: nameValues) {
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> keys;
        Aws::DynamoDB::Model::AttributeValue key;
        key.SetS(name);
        keys.emplace("Name", key);
        table1KeysAndAttributes.AddKeys(keys);
    }

    Aws::Map<Aws::String, Aws::DynamoDB::Model::KeysAndAttributes> requestItems;
    requestItems.emplace(table1Name, table1KeysAndAttributes);

    // Table2: ProductCatalog.
    Aws::String table2Name = "ProductCatalog";
    Aws::DynamoDB::Model::KeysAndAttributes table2KeysAndAttributes;
    table2KeysAndAttributes.SetProjectionExpression("Title, Price, Color");

    // Table2: Set key name, type, and value to search.
    std::vector<Aws::String> idValues = {"102", "103", "201"};
    for (const Aws::String &id: idValues) {
        Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> keys;
        Aws::DynamoDB::Model::AttributeValue key;
        key.SetN(id);
        keys.emplace("Id", key);
        table2KeysAndAttributes.AddKeys(keys);
    }
```

```
   requestItems.emplace(table2Name, table2KeysAndAttributes);

   bool result = true;
   do {  // Use a do loop to handle pagination.
       request.SetRequestItems(requestItems);
       const Aws::DynamoDB::Model::BatchGetItemOutcome &outcome =
dynamoClient.BatchGetItem(
               request);

       if (outcome.IsSuccess()) {
           for (const auto &responsesMapEntry:
outcome.GetResult().GetResponses()) {
               Aws::String tableName = responsesMapEntry.first;
               const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &tableResults = responsesMapEntry.second;
               std::cout << "Retrieved " << tableResults.size()
                         << " responses for table '" << tableName << "'.\n"
                         << std::endl;
               if (tableName == "Forum") {

                   std::cout << "Name | Category | Message | Views" <<
std::endl;
                   for (const Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue> &item: tableResults) {
                       std::cout << item.at("Name").GetS() << " | ";
                       std::cout << item.at("Category").GetS() << " | ";
                       std::cout << (item.count("Message") == 0 ? "" : item.at(
                               "Messages").GetN()) << " | ";
                       std::cout << (item.count("Views") == 0 ? "" : item.at(
                               "Views").GetN()) << std::endl;
                   }
               }
               else {
                   std::cout << "Title | Price | Color" << std::endl;
                   for (const Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue> &item: tableResults) {
                       std::cout << item.at("Title").GetS() << " | ";
                       std::cout << (item.count("Price") == 0 ? "" : item.at(
                               "Price").GetN());
                       if (item.count("Color")) {
                           std::cout << " | ";
                           for (const
std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> &listItem: item.at(
```

```
                                    "Color").GetL())
                                std::cout << listItem->GetS() << " ";
                        }
                        std::cout << std::endl;
                    }
                }
                std::cout << std::endl;
            }

            // If necessary, repeat request for remaining items.
            requestItems = outcome.GetResult().GetUnprocessedKeys();
        }
        else {
            std::cerr << "Batch get item failed: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
            break;
        }
    } while (!requestItems.empty());

    return result;
}
```

- Para obtener información de la API, consulte BatchGetItem en la Referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Recuperación de varios elementos de una tabla

En el siguiente ejemplo `batch-get-items`, se leen varios elementos de la tabla `MusicCollection` mediante un lote de tres solicitudes `GetItem` y se solicita el número de unidades de capacidad de lectura consumidas por la operación. El comando devuelve solo el atributo `AlbumTitle`.

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json \
    --return-consumed-capacity TOTAL
```

Contenido de `request-items.json`:

```
{
    "MusicCollection": {
        "Keys": [
            {
                "Artist": {"S": "No One You Know"},
                "SongTitle": {"S": "Call Me Today"}
            },
            {
                "Artist": {"S": "Acme Band"},
                "SongTitle": {"S": "Happy Day"}
            },
            {
                "Artist": {"S": "No One You Know"},
                "SongTitle": {"S": "Scared of My Shadow"}
            }
        ],
        "ProjectionExpression":"AlbumTitle"
    }
}
```

Salida:

```
{
    "Responses": {
        "MusicCollection": [
            {
                "AlbumTitle": {
                    "S": "Somewhat Famous"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Blue Sky Blues"
                }
            },
            {
                "AlbumTitle": {
                    "S": "Louder Than Ever"
                }
            }
        ]
```

```
    },
    "UnprocessedKeys": {},
    "ConsumedCapacity": [
        {
            "TableName": "MusicCollection",
            "CapacityUnits": 1.5
        }
    ]
}
```

Para obtener más información, consulte Operaciones por lotes en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información de la API, consulte BatchGetItem en la Referencia de comandos de la AWS CLI.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Muestra cómo obtener elementos por lotes mediante el cliente de servicio.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.KeysAndAttributes;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
```

```
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class BatchReadItems {
    public static void main(String[] args){
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """;

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        getBatchItems(dynamoDbClient, tableName);
    }

    public static void getBatchItems(DynamoDbClient dynamoDbClient, String
 tableName) {
        // Define the primary key values for the items you want to retrieve.
        Map<String, AttributeValue> key1 = new HashMap<>();
        key1.put("Artist", AttributeValue.builder().s("Artist1").build());

        Map<String, AttributeValue> key2 = new HashMap<>();
        key2.put("Artist", AttributeValue.builder().s("Artist2").build());

        // Construct the batchGetItem request.
        Map<String, KeysAndAttributes> requestItems = new HashMap<>();
        requestItems.put(tableName, KeysAndAttributes.builder()
            .keys(List.of(key1, key2))
            .projectionExpression("Artist, SongTitle")
            .build());

        BatchGetItemRequest batchGetItemRequest = BatchGetItemRequest.builder()
            .requestItems(requestItems)
            .build();
```

```
        // Make the batchGetItem request.
        BatchGetItemResponse batchGetItemResponse =
 dynamoDbClient.batchGetItem(batchGetItemRequest);

        // Extract and print the retrieved items.
        Map<String, List<Map<String, AttributeValue>>> responses =
 batchGetItemResponse.responses();
        if (responses.containsKey(tableName)) {
            List<Map<String, AttributeValue>> musicItems =
 responses.get(tableName);
            for (Map<String, AttributeValue> item : musicItems) {
                System.out.println("Artist: " + item.get("Artist").s() +
                    ", SongTitle: " + item.get("SongTitle").s());
            }
        } else {
            System.out.println("No items retrieved.");
        }
    }
}
```

Muestra cómo obtener elementos por lotes mediante el cliente de servicio y un paginador.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchGetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.KeysAndAttributes;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class BatchGetItemsPaginator {

    public static void main(String[] args){
        final String usage = """

                Usage:
                    <tableName>

                Where:
```

```
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """";

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        getBatchItemsPaginator(dynamoDbClient, tableName) ;
    }

    public static void getBatchItemsPaginator(DynamoDbClient dynamoDbClient,
 String tableName) {
        // Define the primary key values for the items you want to retrieve.
        Map<String, AttributeValue> key1 = new HashMap<>();
        key1.put("Artist", AttributeValue.builder().s("Artist1").build());

        Map<String, AttributeValue> key2 = new HashMap<>();
        key2.put("Artist", AttributeValue.builder().s("Artist2").build());

        // Construct the batchGetItem request.
        Map<String, KeysAndAttributes> requestItems = new HashMap<>();
        requestItems.put(tableName, KeysAndAttributes.builder()
            .keys(List.of(key1, key2))
            .projectionExpression("Artist, SongTitle")
            .build());

        BatchGetItemRequest batchGetItemRequest = BatchGetItemRequest.builder()
            .requestItems(requestItems)
            .build();

        // Use batchGetItemPaginator for paginated requests.
        dynamoDbClient.batchGetItemPaginator(batchGetItemRequest).stream()
            .flatMap(response -> response.responses().getOrDefault(tableName,
 Collections.emptyList()).stream())
            .forEach(item -> {
                System.out.println("Artist: " + item.get("Artist").s() +
                    ", SongTitle: " + item.get("SongTitle").s());
            });
    }
}
```

- Para obtener información de la API, consulte BatchGetItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener información sobre la API, consulte BatchGet.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { BatchGetCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchGetCommand({
    // Each key in this object is the name of a table. This example refers
    // to a Books table.
    RequestItems: {
      Books: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            Title: "How to AWS",
          },
          {
            Title: "DynamoDB for DBAs",
          },
        ],
        // Only return the "Title" and "PageCount" attributes.
        ProjectionExpression: "Title, PageCount",
      },
    },
```

```
  });

  const response = await docClient.send(command);
  console.log(response.Responses.Books);
  return response;
};
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información de la API, consulte BatchGetItem en la referencia de la API de AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: {
      Keys: [
        { KEY_NAME: { N: "KEY_VALUE_1" } },
        { KEY_NAME: { N: "KEY_VALUE_2" } },
        { KEY_NAME: { N: "KEY_VALUE_3" } },
      ],
      ProjectionExpression: "KEY_NAME, ATTRIBUTE",
    },
  },
};
```

```
ddb.batchGetItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    data.Responses.TABLE_NAME.forEach(function (element, index, array) {
      console.log(element);
    });
  }
});
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información de la API, consulte BatchGetItem en la referencia de la API de AWS SDK para JavaScript.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: obtiene el elemento con el SongTitle «Somewhere Down The Road» de las tablas «Music» y «Songs» de DynamoDB'.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$keysAndAttributes = New-Object Amazon.DynamoDBv2.Model.KeysAndAttributes
$list = New-Object
 'System.Collections.Generic.List[System.Collections.Generic.Dictionary[String,
 Amazon.DynamoDBv2.Model.AttributeValue]]'
$list.Add($key)
$keysAndAttributes.Keys = $list

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
    'Songs' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
}

$batchItems = Get-DDBBatchItem -RequestItem $requestItem
```

```
$batchItems.GetEnumerator() | ForEach-Object {$PSItem.Value} | ConvertFrom-
DDBItem
```

Salida:

```
Name                           Value
----                           -----
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
```

- Para obtener detalles de la API, consulte BatchGetItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: obtiene el elemento con el SongTitle «Somewhere Down The Road» de las tablas «Music» y «Songs» de DynamoDB'.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$keysAndAttributes = New-Object Amazon.DynamoDBv2.Model.KeysAndAttributes
$list = New-Object
 'System.Collections.Generic.List[System.Collections.Generic.Dictionary[String,
 Amazon.DynamoDBv2.Model.AttributeValue]]'
$list.Add($key)
$keysAndAttributes.Keys = $list

$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
    'Songs' = [Amazon.DynamoDBv2.Model.KeysAndAttributes]$keysAndAttributes
```

```
}

$batchItems = Get-DDBBatchItem -RequestItem $requestItem
$batchItems.GetEnumerator() | ForEach-Object {$PSItem.Value} | ConvertFrom-
DDBItem
```

Salida:

```
Name                           Value
----                           -----
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
Artist                         No One You Know
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
CriticRating                   10
Genre                          Country
Price                          1.94
```

- Para obtener detalles de la API, consulte BatchGetItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import decimal
import json
import logging
import os
import pprint
```

```
import time
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
dynamodb = boto3.resource("dynamodb")

MAX_GET_SIZE = 100  # Amazon DynamoDB rejects a get batch larger than 100 items.


def do_batch_get(batch_keys):
    """
    Gets a batch of items from Amazon DynamoDB. Batches can contain keys from
    more than one table.

    When Amazon DynamoDB cannot process all items in a batch, a set of
 unprocessed
    keys is returned. This function uses an exponential backoff algorithm to
 retry
    getting the unprocessed keys until all are retrieved or the specified
    number of tries is reached.

    :param batch_keys: The set of keys to retrieve. A batch can contain at most
 100
                       keys. Otherwise, Amazon DynamoDB returns an error.
    :return: The dictionary of retrieved items grouped under their respective
             table names.
    """
    tries = 0
    max_tries = 5
    sleepy_time = 1  # Start with 1 second of sleep, then exponentially increase.
    retrieved = {key: [] for key in batch_keys}
    while tries < max_tries:
        response = dynamodb.batch_get_item(RequestItems=batch_keys)
        # Collect any retrieved items and retry unprocessed keys.
        for key in response.get("Responses", []):
            retrieved[key] += response["Responses"][key]
        unprocessed = response["UnprocessedKeys"]
        if len(unprocessed) > 0:
            batch_keys = unprocessed
            unprocessed_count = sum(
                [len(batch_key["Keys"]) for batch_key in batch_keys.values()]
            )
            logger.info(
```

```
                "%s unprocessed keys returned. Sleep, then retry.",
 unprocessed_count
            )
            tries += 1
            if tries < max_tries:
                logger.info("Sleeping for %s seconds.", sleepy_time)
                time.sleep(sleepy_time)
                sleepy_time = min(sleepy_time * 2, 32)
        else:
            break

    return retrieved
```

- Para obtener información sobre la API, consulte BatchGetItem en la referencia de la API de AWS SDK para Python (Boto3).

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB


    /// Gets an array of `Movie` objects describing all the movies in the
    /// specified list. Any movies that aren't found in the list have no
    /// corresponding entry in the resulting array.
    ///
    /// - Parameters
    ///     - keys: An array of tuples, each of which specifies the title and
    ///       release year of a movie to fetch from the table.
    ///
    /// - Returns:
```

```
    ///     - An array of `Movie` objects describing each match found in the
    ///     table.
    ///
    /// - Throws:
    ///     - `MovieError.ClientUninitialized` if the DynamoDB client has not
    ///     been initialized.
    ///     - DynamoDB errors are thrown without change.
    func batchGet(keys: [(title: String, year: Int)]) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MovieError.ClientUninitialized
            }

            var movieList: [Movie] = []
            var keyItems: [[Swift.String: DynamoDBClientTypes.AttributeValue]] =
[]

            // Convert the list of keys into the form used by DynamoDB.

            for key in keys {
                let item: [Swift.String: DynamoDBClientTypes.AttributeValue] = [
                    "title": .s(key.title),
                    "year": .n(String(key.year))
                ]
                keyItems.append(item)
            }

            // Create the input record for `batchGetItem()`. The list of
requested
            // items is in the `requestItems` property. This array contains one
            // entry for each table from which items are to be fetched. In this
            // example, there's only one table containing the movie data.
            //
            // If we wanted this program to also support searching for matches
            // in a table of book data, we could add a second `requestItem`
            // mapping the name of the book table to the list of items we want to
            // find in it.
            let input = BatchGetItemInput(
                requestItems: [
                    self.tableName: .init(
                        consistentRead: true,
                        keys: keyItems
                    )
                ]
```

```
            )

            // Fetch the matching movies from the table.

            let output = try await client.batchGetItem(input: input)

            // Get the set of responses. If there aren't any, return the empty
            // movie list.

            guard let responses = output.responses else {
                return movieList
            }

            // Get the list of matching items for the table with the name
            // `tableName`.

            guard let responseList = responses[self.tableName] else {
                return movieList
            }

            // Create `Movie` items for each of the matching movies in the table
            // and add them to the `MovieList` array.

            for response in responseList {
                try movieList.append(Movie(withItem: response))
            }

            return movieList
        } catch {
            print("ERROR: batchGet", dump(error))
            throw error
        }
    }
```

- Para obtener detalles sobre la API, consulte BatchGetItem en la Referencia de la API del AWS SDK para Swift.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de **`BatchWriteItem`** con un SDK de AWS o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `BatchWriteItem`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- Conceptos básicos

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Escribe un lote de elementos en la tabla de películas.

```
/// <summary>
/// Loads the contents of a JSON file into a list of movies to be
/// added to the DynamoDB table.
/// </summary>
/// <param name="movieFileName">The name of the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public List<Movie> ImportMovies(string movieFileName)
{
    var moviesList = new List<Movie>();
    if (!File.Exists(movieFileName))
    {
        return moviesList;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonSerializer.Deserialize<List<Movie>>(
        json,
        new JsonSerializerOptions
        {
```

```
                PropertyNameCaseInsensitive = true
            });

        // Now return the first 250 entries.
        if (allMovies != null && allMovies.Any())
        {
            moviesList = allMovies.GetRange(0, 250);
        }
        return moviesList;
    }

    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie
data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of
IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);
```

```
            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch
write operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

- Para obtener información sobre la API, consulte BatchWriteItem en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_batch_write_item
#
# This function writes a batch of items into a DynamoDB table.
#
# Parameters:
#       -i item  -- Path to json file containing the items to write.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_batch_write_item() {
  local item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_batch_write_item"
    echo "Write a batch of items into a DynamoDB table."
    echo " -i item  -- Path to json file containing the items to write."
    echo ""
  }
  while getopts "i:h" option; do
    case "${option}" in
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
```

```
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""

  response=$(aws dynamodb batch-write-item \
    --request-items file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  return 0
}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
```

```
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información de la API, consulte BatchWriteItem en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Batch write items from a JSON file.
/*!
  \sa batchWriteItem()
  \param jsonFilePath: JSON file path.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

/*
 * The input for this routine is a JSON file that you can download from the
 following URL:
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
SampleData.html.
 *
 * The JSON data uses the BatchWriteItem API request syntax. The JSON strings are
 * converted to AttributeValue objects. These AttributeValue objects will then
 generate
 * JSON strings when constructing the BatchWriteItem request, essentially
 outputting
 * their input.
 *
 * This is perhaps an artificial example, but it demonstrates the APIs.
 */

bool AwsDoc::DynamoDB::batchWriteItem(const Aws::String &jsonFilePath,
                                      const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    std::ifstream fileStream(jsonFilePath);
```

```
   if (!fileStream) {
       std::cerr << "Error: could not open file '" << jsonFilePath << "'."
                 << std::endl;
   }

   std::stringstream stringStream;
   stringStream << fileStream.rdbuf();
   Aws::Utils::Json::JsonValue jsonValue(stringStream);

   Aws::DynamoDB::Model::BatchWriteItemRequest batchWriteItemRequest;
   Aws::Map<Aws::String, Aws::Utils::Json::JsonView> level1Map =
jsonValue.View().GetAllObjects();
   for (const auto &level1Entry: level1Map) {
       const Aws::Utils::Json::JsonView &entriesView = level1Entry.second;
       const Aws::String &tableName = level1Entry.first;
       // The JSON entries at this level are as follows:
       //  key - table name
       //  value - list of request objects
       if (!entriesView.IsListType()) {
           std::cerr << "Error: JSON file entry '"
                     << tableName << "' is not a list." << std::endl;
           continue;
       }

       Aws::Utils::Array<Aws::Utils::Json::JsonView> entries =
entriesView.AsArray();

       Aws::Vector<Aws::DynamoDB::Model::WriteRequest> writeRequests;
       if (AwsDoc::DynamoDB::addWriteRequests(tableName, entries,
                                              writeRequests)) {
           batchWriteItemRequest.AddRequestItems(tableName, writeRequests);
       }
   }

   Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

   Aws::DynamoDB::Model::BatchWriteItemOutcome outcome =
dynamoClient.BatchWriteItem(
           batchWriteItemRequest);

   if (outcome.IsSuccess()) {
       std::cout << "DynamoDB::BatchWriteItem was successful." << std::endl;
   }
```

```
    else {
        std::cerr << "Error with DynamoDB::BatchWriteItem. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return outcome.IsSuccess();
}

//! Convert requests in JSON format to a vector of WriteRequest objects.
/*!
  \sa addWriteRequests()
  \param tableName: Name of the table for the write operations.
  \param requestsJson: Request data in JSON format.
  \param writeRequests: Vector to receive the WriteRequest objects.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::addWriteRequests(const Aws::String &tableName,
                                        const
 Aws::Utils::Array<Aws::Utils::Json::JsonView> &requestsJson,

 Aws::Vector<Aws::DynamoDB::Model::WriteRequest> &writeRequests) {
    for (size_t i = 0; i < requestsJson.GetLength(); ++i) {
        const Aws::Utils::Json::JsonView &requestsEntry = requestsJson[i];
        if (!requestsEntry.IsObject()) {
            std::cerr << "Error: incorrect requestsEntry type "
                      << requestsEntry.WriteReadable() << std::endl;
            return false;
        }

        Aws::Map<Aws::String, Aws::Utils::Json::JsonView> requestsMap =
 requestsEntry.GetAllObjects();

        for (const auto &request: requestsMap) {
            const Aws::String &requestType = request.first;
            const Aws::Utils::Json::JsonView &requestJsonView = request.second;

            if (requestType == "PutRequest") {
                if (!requestJsonView.ValueExists("Item")) {
                    std::cerr << "Error: item key missing for requests "
                              << requestJsonView.WriteReadable() << std::endl;
                    return false;
                }
```

```
                Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 attributes;
                if (!getAttributeObjectsMap(requestJsonView.GetObject("Item"),
                                            attributes)) {
                    std::cerr << "Error getting attributes "
                              << requestJsonView.WriteReadable() << std::endl;
                    return false;
                }

                Aws::DynamoDB::Model::PutRequest putRequest;
                putRequest.SetItem(attributes);
                writeRequests.push_back(
                        Aws::DynamoDB::Model::WriteRequest().WithPutRequest(
                                putRequest));
            }
            else {
                std::cerr << "Error: unimplemented request type '" << requestType
                          << "'." << std::endl;
            }
        }
    }

    return true;
}

//! Generate a map of AttributeValue objects from JSON records.
/*!
  \sa getAttributeObjectsMap()
  \param jsonView: JSONView of attribute records.
  \param writeRequests: Map to receive the AttributeValue objects.
  \return bool: Function succeeded.
 */
bool
AwsDoc::DynamoDB::getAttributeObjectsMap(const Aws::Utils::Json::JsonView
 &jsonView,
                                         Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue> &attributes) {
    Aws::Map<Aws::String, Aws::Utils::Json::JsonView> objectsMap =
 jsonView.GetAllObjects();
    for (const auto &entry: objectsMap) {
        const Aws::String &attributeKey = entry.first;
        const Aws::Utils::Json::JsonView &attributeJsonView = entry.second;

        if (!attributeJsonView.IsObject()) {
```

```
            std::cerr << "Error: attribute not an object "
                      << attributeJsonView.WriteReadable() << std::endl;
            return false;
        }

        attributes.emplace(attributeKey,

 Aws::DynamoDB::Model::AttributeValue(attributeJsonView));
    }

    return true;
}
```

- Para obtener información sobre la API, consulte BatchWriteItem en la referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Adición de varios elementos a una tabla

En el siguiente ejemplo de `batch-write-item`, se añaden tres elementos nuevos a la tabla `MusicCollection` mediante un lote de tres solicitudes `PutItem`. También solicita información sobre el número de unidades de capacidad de escritura consumidas por la operación y cualquier colección de elementos modificada por la operación.

```
aws dynamodb batch-write-item \
    --request-items file://request-items.json \
    --return-consumed-capacity INDEXES \
    --return-item-collection-metrics SIZE
```

Contenido de `request-items.json`:

```
{
    "MusicCollection": [
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "No One You Know"},
```

```
                    "SongTitle": {"S": "Call Me Today"},
                    "AlbumTitle": {"S": "Somewhat Famous"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "Acme Band"},
                    "SongTitle": {"S": "Happy Day"},
                    "AlbumTitle": {"S": "Songs About Life"}
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Artist": {"S": "No One You Know"},
                    "SongTitle": {"S": "Scared of My Shadow"},
                    "AlbumTitle": {"S": "Blue Sky Blues"}
                }
            }
        }
    ]
}
```

Salida:

```
{
    "UnprocessedItems": {},
    "ItemCollectionMetrics": {
        "MusicCollection": [
            {
                "ItemCollectionKey": {
                    "Artist": {
                        "S": "No One You Know"
                    }
                },
                "SizeEstimateRangeGB": [
                    0.0,
                    1.0
                ]
            },
```

```
            {
                "ItemCollectionKey": {
                    "Artist": {
                        "S": "Acme Band"
                    }
                },
                "SizeEstimateRangeGB": [
                    0.0,
                    1.0
                ]
            }
        ]
    },
    "ConsumedCapacity": [
        {
            "TableName": "MusicCollection",
            "CapacityUnits": 6.0,
            "Table": {
                "CapacityUnits": 3.0
            },
            "LocalSecondaryIndexes": {
                "AlbumTitleIndex": {
                    "CapacityUnits": 3.0
                }
            }
        }
    ]
}
```

Para obtener más información, consulte Operaciones por lotes en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información de la API, consulte BatchWriteItem en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovieBatch adds a slice of movies to the DynamoDB table. The function sends
// batches of 25 movies to DynamoDB until all movies are added or it reaches the
// specified maximum.
func (basics TableBasics) AddMovieBatch(ctx context.Context, movies []Movie,
 maxMovies int) (int, error) {
 var err error
 var item map[string]types.AttributeValue
```

```
 written := 0
 batchSize := 25 // DynamoDB allows a maximum batch size of 25 items.
 start := 0
 end := start + batchSize
 for start < maxMovies && start < len(movies) {
  var writeReqs []types.WriteRequest
  if end > len(movies) {
   end = len(movies)
  }
  for _, movie := range movies[start:end] {
   item, err = attributevalue.MarshalMap(movie)
   if err != nil {
    log.Printf("Couldn't marshal movie %v for batch writing. Here's why: %v\n",
 movie.Title, err)
   } else {
    writeReqs = append(
     writeReqs,
     types.WriteRequest{PutRequest: &types.PutRequest{Item: item}},
    )
   }
  }
  _, err = basics.DynamoDbClient.BatchWriteItem(ctx,
 &dynamodb.BatchWriteItemInput{
   RequestItems: map[string][]types.WriteRequest{basics.TableName: writeReqs}})
  if err != nil {
   log.Printf("Couldn't add a batch of movies to %v. Here's why: %v\n",
 basics.TableName, err)
  } else {
   written += len(writeReqs)
  }
  start = end
  end += batchSize
 }

 return written, err
}
```

Defina una estructura Película para utilizar en este ejemplo.

```
import (
```

```
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obtener información sobre la API, consulte BatchWriteItem en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Inserta muchos elementos en una tabla con el cliente de servicio.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.BatchWriteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.BatchWriteItemResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutRequest;
import software.amazon.awssdk.services.dynamodb.model.WriteRequest;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class BatchWriteItems {
    public static void main(String[] args){
        final String usage = """
```

```
                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music).\s
                """;

        String tableName = "Music";
        Region region = Region.US_EAST_1;
        DynamoDbClient dynamoDbClient = DynamoDbClient.builder()
            .region(region)
            .build();

        addBatchItems(dynamoDbClient, tableName);
    }

    public static void addBatchItems(DynamoDbClient dynamoDbClient, String
tableName) {
        // Specify the updates you want to perform.
        List<WriteRequest> writeRequests = new ArrayList<>();

        // Set item 1.
        Map<String, AttributeValue> item1Attributes = new HashMap<>();
        item1Attributes.put("Artist",
AttributeValue.builder().s("Artist1").build());
        item1Attributes.put("Rating", AttributeValue.builder().s("5").build());
        item1Attributes.put("Comments", AttributeValue.builder().s("Great
song!").build());
        item1Attributes.put("SongTitle",
AttributeValue.builder().s("SongTitle1").build());

writeRequests.add(WriteRequest.builder().putRequest(PutRequest.builder().item(item1Attri

        // Set item 2.
        Map<String, AttributeValue> item2Attributes = new HashMap<>();
        item2Attributes.put("Artist",
AttributeValue.builder().s("Artist2").build());
        item2Attributes.put("Rating", AttributeValue.builder().s("4").build());
        item2Attributes.put("Comments", AttributeValue.builder().s("Nice
melody.").build());
        item2Attributes.put("SongTitle",
AttributeValue.builder().s("SongTitle2").build());
```

```
 writeRequests.add(WriteRequest.builder().putRequest(PutRequest.builder().item(item2Attri

        try {
            // Create the BatchWriteItemRequest.
            BatchWriteItemRequest batchWriteItemRequest =
 BatchWriteItemRequest.builder()
                .requestItems(Map.of(tableName, writeRequests))
                .build();

            // Execute the BatchWriteItem operation.
            BatchWriteItemResponse batchWriteItemResponse =
 dynamoDbClient.batchWriteItem(batchWriteItemRequest);

            // Process the response.
            System.out.println("Batch write successful: " +
 batchWriteItemResponse);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

Inserta muchos elementos en una tabla mediante el cliente mejorado.

```
import com.example.dynamodb.Customer;
import com.example.dynamodb.Music;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbEnhancedClient;
import software.amazon.awssdk.enhanced.dynamodb.DynamoDbTable;
import software.amazon.awssdk.enhanced.dynamodb.Key;
import software.amazon.awssdk.enhanced.dynamodb.TableSchema;
import
 software.amazon.awssdk.enhanced.dynamodb.model.BatchWriteItemEnhancedRequest;
import software.amazon.awssdk.enhanced.dynamodb.model.WriteBatch;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
```

```
import java.time.ZoneOffset;

/*
 * Before running this code example, create an Amazon DynamoDB table named
 Customer with these columns:
 *   - id - the id of the record that is the key
 *   - custName - the customer name
 *   - email - the email value
 *   - registrationDate - an instant value when the item was added to the table
 *
 * Also, ensure that you have set up your development environment, including your
 credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class EnhancedBatchWriteItems {
        public static void main(String[] args) {
                Region region = Region.US_EAST_1;
                DynamoDbClient ddb = DynamoDbClient.builder()
                                .region(region)
                                .build();
                DynamoDbEnhancedClient enhancedClient =
 DynamoDbEnhancedClient.builder()
                                .dynamoDbClient(ddb)
                                .build();
                putBatchRecords(enhancedClient);
                ddb.close();
        }

        public static void putBatchRecords(DynamoDbEnhancedClient enhancedClient)
 {
                try {
                        DynamoDbTable<Customer> customerMappedTable =
 enhancedClient.table("Customer",
                                        TableSchema.fromBean(Customer.class));
                        DynamoDbTable<Music> musicMappedTable =
 enhancedClient.table("Music",
                                        TableSchema.fromBean(Music.class));
                        LocalDate localDate = LocalDate.parse("2020-04-07");
                        LocalDateTime localDateTime = localDate.atStartOfDay();
```

```
                        Instant instant =
localDateTime.toInstant(ZoneOffset.UTC);

                        Customer record2 = new Customer();
                        record2.setCustName("Fred Pink");
                        record2.setId("id110");
                        record2.setEmail("fredp@noserver.com");
                        record2.setRegistrationDate(instant);

                        Customer record3 = new Customer();
                        record3.setCustName("Susan Pink");
                        record3.setId("id120");
                        record3.setEmail("spink@noserver.com");
                        record3.setRegistrationDate(instant);

                        Customer record4 = new Customer();
                        record4.setCustName("Jerry orange");
                        record4.setId("id101");
                        record4.setEmail("jorange@noserver.com");
                        record4.setRegistrationDate(instant);

                        BatchWriteItemEnhancedRequest
batchWriteItemEnhancedRequest = BatchWriteItemEnhancedRequest
                                        .builder()
                                        .writeBatches(

WriteBatch.builder(Customer.class) // add items to the Customer

         // table

.mappedTableResource(customerMappedTable)

.addPutItem(builder -> builder.item(record2))

.addPutItem(builder -> builder.item(record3))

.addPutItem(builder -> builder.item(record4))
                                                                        .build(),

WriteBatch.builder(Music.class) // delete an item from the Music

      // table

.mappedTableResource(musicMappedTable)
```

```
 .addDeleteItem(builder -> builder.key(

       Key.builder().partitionValue(

                        "Famous Band")

                        .build()))
                                                                    .build())
                                      .build();

                         // Add three items to the Customer table and delete one
 item from the Music
                         // table.

 enhancedClient.batchWriteItem(batchWriteItemEnhancedRequest);
                         System.out.println("done");

                } catch (DynamoDbException e) {
                        System.err.println(e.getMessage());
                        System.exit(1);
                }
        }
}
```

- Para obtener información sobre la API, consulte BatchWriteItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener información sobre la API, consulte BatchWrite.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";
import { readFileSync } from "node:fs";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const file = readFileSync(
    `${dirname}../../../../../resources/sample_files/movies.json`,
  );

  const movies = JSON.parse(file.toString());

  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);

  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
        // An existing table is required. A composite key of 'title' and 'year'
 is recommended
        // to account for duplicate titles.
        BatchWriteMoviesTable: putRequests,
```

```
      },
    });

    await docClient.send(command);
  }
};
```

- Para obtener información sobre la API, consulte BatchWriteItem en la referencia de la API de AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  RequestItems: {
    TABLE_NAME: [
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            KEY: { N: "KEY_VALUE" },
```

```
            ATTRIBUTE_1: { S: "ATTRIBUTE_1_VALUE" },
            ATTRIBUTE_2: { N: "ATTRIBUTE_2_VALUE" },
          },
        },
      },
    ],
  },
};

ddb.batchWriteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte BatchWriteItem en la referencia de la API de AWS SDK para JavaScript.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    public function writeBatch(string $TableName, array $Batch, int $depth = 2)
    {
        if (--$depth <= 0) {
            throw new Exception("Max depth exceeded. Please try with fewer batch
 items or increase depth.");
        }

        $marshal = new Marshaler();
```

```
        $total = 0;
        foreach (array_chunk($Batch, 25) as $Items) {
            foreach ($Items as $Item) {
                $BatchWrite['RequestItems'][$TableName][] = ['PutRequest' =>
 ['Item' => $marshal->marshalItem($Item)]];
            }
            try {
                echo "Batching another " . count($Items) . " for a total of " .
 ($total += count($Items)) . " items!\n";
                $response = $this->dynamoDbClient->batchWriteItem($BatchWrite);
                $BatchWrite = [];
            } catch (Exception $e) {
                echo "uh oh...";
                echo $e->getMessage();
                die();
            }
            if ($total >= 250) {
                echo "250 movies is probably enough. Right? We can stop there.
\n";
                break;
            }
        }
    }
```

- Para obtener información sobre la API, consulte BatchWriteItem en la referencia de la API de AWS SDK para PHP.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: crea un nuevo elemento o sustituye un elemento existente por uno nuevo en las tablas Music y Songs de DynamoDB.

```
$item = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 10.0
```

```
} | ConvertTo-DDBItem

$writeRequest = New-Object Amazon.DynamoDBv2.Model.WriteRequest
$writeRequest.PutRequest = [Amazon.DynamoDBv2.Model.PutRequest]$item
```

Salida:

```
$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- Para obtener detalles de la API, consulte BatchWriteItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: crea un nuevo elemento o sustituye un elemento existente por uno nuevo en las tablas Music y Songs de DynamoDB.

```
$item = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 10.0
} | ConvertTo-DDBItem

$writeRequest = New-Object Amazon.DynamoDBv2.Model.WriteRequest
$writeRequest.PutRequest = [Amazon.DynamoDBv2.Model.PutRequest]$item
```

Salida:

```
$requestItem = @{
    'Music' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
    'Songs' = [Amazon.DynamoDBv2.Model.WriteRequest]($writeRequest)
}

Set-DDBBatchItem -RequestItem $requestItem
```

- Para obtener detalles de la API, consulte BatchWriteItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
```

```
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def write_batch(self, movies):
        """
        Fills an Amazon DynamoDB table with the specified data, using the Boto3
        Table.batch_writer() function to put the items in the table.
        Inside the context manager, Table.batch_writer builds a list of
        requests. On exiting the context manager, Table.batch_writer starts
sending
        batches of write requests to Amazon DynamoDB and automatically
        handles chunking, buffering, and retrying.

        :param movies: The data to put in the table. Each item must contain at
least
                       the keys required by the schema that was specified when
the
                       table was created.
        """
        try:
            with self.table.batch_writer() as writer:
                for movie in movies:
                    writer.put_item(Item=movie)
        except ClientError as err:
            logger.error(
                "Couldn't load data into table %s. Here's why: %s: %s",
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Para obtener información sobre la API, consulte BatchWriteItem en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Fills an Amazon DynamoDB table with the specified data. Items are sent in
  # batches of 25 until all items are written.
  #
  # @param movies [Enumerable] The data to put in the table. Each item must
 contain at least
  #                            the keys required by the schema that was specified
 when the
  #                            table was created.
  def write_batch(movies)
    index = 0
    slice_size = 25
    while index < movies.length
      movie_items = []
      movies[index, slice_size].each do |movie|
        movie_items.append({ put_request: { item: movie } })
      end
      @dynamo_resource.client.batch_write_item({ request_items: { @table.name =>
 movie_items } })
      index += slice_size
    end
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts(
      "Couldn't load data into table #{@table.name}. Here's why:"
```

```
    )
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Para obtener información sobre la API, consulte BatchWriteItem en la referencia de la API de AWS SDK para Ruby.

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB


    /// Populate the movie database from the specified JSON file.
    ///
    /// - Parameter jsonPath: Path to a JSON file containing movie data.
    ///
    func populate(jsonPath: String) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Create a Swift `URL` and use it to load the file into a `Data`
            // object. Then decode the JSON into an array of `Movie` objects.

            let fileUrl = URL(fileURLWithPath: jsonPath)
            let jsonData = try Data(contentsOf: fileUrl)

            var movieList = try JSONDecoder().decode([Movie].self, from:
 jsonData)
```

```
            // Truncate the list to the first 200 entries or so for this example.

            if movieList.count > 200 {
                movieList = Array(movieList[...199])
            }

            // Before sending records to the database, break the movie list into
            // 25-entry chunks, which is the maximum size of a batch item
request.

            let count = movieList.count
            let chunks = stride(from: 0, to: count, by: 25).map {
                Array(movieList[$0 ..< Swift.min($0 + 25, count)])
            }

            // For each chunk, create a list of write request records and
populate
            // them with `PutRequest` requests, each specifying one movie from
the
            // chunk. Once the chunk's items are all in the `PutRequest` list,
            // send them to Amazon DynamoDB using the
            // `DynamoDBClient.batchWriteItem()` function.

            for chunk in chunks {
                var requestList: [DynamoDBClientTypes.WriteRequest] = []

                for movie in chunk {
                    let item = try await movie.getAsItem()
                    let request = DynamoDBClientTypes.WriteRequest(
                        putRequest: .init(
                            item: item
                        )
                    )
                    requestList.append(request)
                }

                let input = BatchWriteItemInput(requestItems: [tableName:
requestList])
                _ = try await client.batchWriteItem(input: input)
            }
        } catch {
            print("ERROR: populate:", dump(error))
            throw error
        }
```

```
    }
```

- Para obtener detalles de la API, consulte BatchWriteItem en la referencia de la API del SDK de AWS para Swift.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de **`CreateTable`** con un SDK de AWS o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `CreateTable`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Conceptos básicos
- Aceleración de lecturas con DAX
- Escenarios avanzados de índice secundario global
- Creación de una tabla con índice secundario global
- Creación de una tabla con rendimiento en caliente habilitado
- Creación y administración de tablas globales que muestren MREC
- Creación y administración de tablas globales de MRSC
- Administración de índices secundarios globales
- Administración de las políticas basadas en recursos
- Configuración del control de acceso basado en atributos
- Trabajo con tablas globales y coherencia final de replicación de varias regiones (MREC)
- Trabajo con índices secundarios locales
- Trabajo con el etiquetado de recursos
- Trabajo con Streams y tiempo de vida
- Trabajo con el cifrado de tablas

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
 CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
```

```
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };

            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
 table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table
 {tableName}. {ex.Message}");
            throw;
        }
    }
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_create_table
#
# This function creates an Amazon DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to create.
#       -a attribute_definitions -- JSON file path of a list of attributes and
 their types.
#       -k key_schema -- JSON file path of a list of attributes and their key
 types.
#
```

```
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_create_table() {
  local table_name attribute_definitions key_schema response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_create_table"
    echo "Creates an Amazon DynamoDB table with on-demand billing."
    echo " -n table_name  -- The name of the table to create."
    echo " -a attribute_definitions -- JSON file path of a list of attributes and
 their types."
    echo " -k key_schema -- JSON file path of a list of attributes and their key
 types."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:a:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      a) attribute_definitions="${OPTARG}" ;;
      k) key_schema="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
```

```
    return 1
  fi

  if [[ -z "$attribute_definitions" ]]; then
    errecho "ERROR: You must provide an attribute definitions json file path the
 -a parameter."
    usage
    return 1
  fi

  if [[ -z "$key_schema" ]]; then
    errecho "ERROR: You must provide a key schema json file path the -k
 parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    attribute_definitions:   $attribute_definitions"
  iecho "    key_schema:   $key_schema"
  iecho ""

  response=$(aws dynamodb create-table \
    --table-name "$table_name" \
    --attribute-definitions file://"$attribute_definitions" \
    --billing-mode PAY_PER_REQUEST \
    --key-schema file://"$key_schema" )

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports create-table operation failed.$response"
    return 1
  fi

  return 0
}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
##############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
##############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

##############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
##############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

##############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
##############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
```

```
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información sobre la API, consulte CreateTable en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Create an Amazon DynamoDB table.
/*!
  \sa createTable()
  \param tableName: Name for the DynamoDB table.
  \param primaryKey: Primary key for the DynamoDB table.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::createTable(const Aws::String &tableName,
                                   const Aws::String &primaryKey,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
```

```
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::cout << "Creating table " << tableName <<
              " with a simple primary key: \"" << primaryKey << "\"." <<
 std::endl;

    Aws::DynamoDB::Model::CreateTableRequest request;

    Aws::DynamoDB::Model::AttributeDefinition hashKey;
    hashKey.SetAttributeName(primaryKey);
    hashKey.SetAttributeType(Aws::DynamoDB::Model::ScalarAttributeType::S);
    request.AddAttributeDefinitions(hashKey);

    Aws::DynamoDB::Model::KeySchemaElement keySchemaElement;
    keySchemaElement.WithAttributeName(primaryKey).WithKeyType(
            Aws::DynamoDB::Model::KeyType::HASH);
    request.AddKeySchema(keySchemaElement);

    Aws::DynamoDB::Model::ProvisionedThroughput throughput;
    throughput.WithReadCapacityUnits(5).WithWriteCapacityUnits(5);
    request.SetProvisionedThroughput(throughput);
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::CreateTableOutcome &outcome =
 dynamoClient.CreateTable(
              request);
    if (outcome.IsSuccess()) {
        std::cout << "Table \""
                  << outcome.GetResult().GetTableDescription().GetTableName() <<
                  " created!" << std::endl;
    }
    else {
        std::cerr << "Failed to create table: " <<
 outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Código que espera a que se active la tabla.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obtener información sobre la API, consulte CreateTable en la Referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Ejemplo 1: Creación de una tabla con etiquetas

En el siguiente ejemplo `create-table`, se utilizan los atributos y el esquema de claves especificados para crear una tabla denominada `MusicCollection`. Esta tabla utiliza el rendimiento aprovisionado y se cifrará en reposo con la CMK predeterminada propiedad de AWS. El comando también aplica una etiqueta a la tabla, con una clave de `Owner` y un valor de `blueTeam`.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Owner,Value=blueTeam
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
```

```
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "CREATING",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": "2020-05-26T16:04:41.627000-07:00",
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    }
}
```

Para obtener más información, consulte Operaciones básicas con tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 2: Creación de una tabla en modo bajo demanda

En el siguiente ejemplo, se crea una tabla denominada `MusicCollection` mediante el modo bajo demanda, en lugar del modo de rendimiento aprovisionado. Esto resulta útil para tablas con cargas de trabajo impredecibles.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:44:10.807000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 0,
            "WriteCapacityUnits": 0
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "BillingModeSummary": {
            "BillingMode": "PAY_PER_REQUEST"
        }
    }
}
```

Para obtener más información, consulte Operaciones básicas con tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 3: creación de una tabla y cifrarla con una CMK administrada por el cliente

En el siguiente ejemplo, se crea una tabla denominada `MusicCollection` y se cifra mediante una CMK administrada por el cliente.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-
abcd-1234-a123-ab1234a1b234
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T11:12:16.431000-07:00",
```

```
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "SSEDescription": {
            "Status": "ENABLED",
            "SSEType": "KMS",
            "KMSMasterKeyArn": "arn:aws:kms:us-west-2:123456789012:key/abcd1234-
abcd-1234-a123-ab1234a1b234"
        }
    }
}
```

Para obtener más información, consulte Operaciones básicas con tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 4: creación de una tabla con un índice secundario local

En el siguiente ejemplo, se utilizan los atributos y el esquema de claves especificados para crear una tabla denominada `MusicCollection` con un índice secundario local denominado `AlbumTitleIndex`.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-
definitions AttributeName=Artist,AttributeType=S AttributeName=SongTitle,AttributeType=S
 \
    --key-
schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\": \"Artist\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\": \"AlbumTitle\",\"KeyType\":\"RANGE\"}
```

```
                ],
                \"Projection\": {
                    \"ProjectionType\": \"INCLUDE\",
                    \"NonKeyAttributes\": [\"Genre\", \"Year\"]
                }
            }
        ]"
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
```

```
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Genre",
                        "Year"
                    ]
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ]
    }
}
```

Para obtener más información, consulte Operaciones básicas con tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 5: creación de una tabla con un índice secundario global

En el siguiente ejemplo, se crea una tabla llamada `GameScores` con un índice secundario global denominado `GameTitleIndex`. La tabla base tiene una clave de partición de `UserId` y una clave de ordenación de `GameTitle`, lo que le permite encontrar eficientemente la mejor

puntuación de un usuario individual para un juego específico, mientras que el GSI tiene una clave de partición de `GameTitle` y una clave de ordenación de `TopScore`, lo que te permite encontrar rápidamente la puntuación más alta en general para un juego en particular.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-schema AttributeName=UserId,KeyType=HASH \
                 AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GameTitleIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"GameTitle\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"TopScore\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {
                    \"ProjectionType\":\"INCLUDE\",
                    \"NonKeyAttributes\":[\"UserId\"]
                },
                \"ProvisionedThroughput\": {
                    \"ReadCapacityUnits\": 10,
                    \"WriteCapacityUnits\": 5
                }
            }
        ]"
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
```

```
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-26T17:28:15.602000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
```

```
                        "UserId"
                    ]
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            }
        ]
    }
}
```

Para obtener más información, consulte Operaciones básicas con tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 6: creación de una tabla con varios índices secundarios globales a la vez

En el siguiente ejemplo, se crea una tabla denominada `GameScores` con dos índices secundarios globales. Los esquemas GSI se transfieren mediante un archivo, en lugar de hacerlo a través de la línea de comandos.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --global-secondary-indexes file://gsi.json
```

Contenido de `gsi.json`:

```
[
    {
        "IndexName": "GameTitleIndex",
```

```
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "TopScore",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        }
    },
    {
        "IndexName": "GameDateIndex",
        "KeySchema": [
            {
                "AttributeName": "GameTitle",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "Date",
                "KeyType": "RANGE"
            }
        ],
        "Projection": {
            "ProjectionType": "ALL"
        },
        "ProvisionedThroughput": {
            "ReadCapacityUnits": 5,
            "WriteCapacityUnits": 5
        }
    }
]
```

Salida:

```
{
```

```
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Date",
                "AttributeType": "S"
            },
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "TopScore",
                "AttributeType": "N"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-08-04T16:40:55.524000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "GameTitleIndex",
```

```
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "TopScore",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 5
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameTitleIndex"
            },
            {
                "IndexName": "GameDateIndex",
                "KeySchema": [
                    {
                        "AttributeName": "GameTitle",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "Date",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
```

```
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/index/GameDateIndex"
            }
        ]
    }
}
```

Para obtener más información, consulte Operaciones básicas con tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 7: creación de una tabla que tiene habilitado Streams

En el siguiente ejemplo, se crea una tabla denominada `GameScores` con DynamoDB Streams habilitado. En el flujo se escribirán tanto las imágenes nuevas como las antiguas de cada elemento.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=NEW_AND_OLD_IMAGES
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
```

```
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2020-05-27T10:49:34.056000-07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_AND_OLD_IMAGES"
        },
        "LatestStreamLabel": "2020-05-27T17:49:34.056",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2020-05-27T17:49:34.056"
    }
}
```

Para obtener más información, consulte Operaciones básicas con tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 8: creación de una tabla con un flujo habilitado solo de claves

En el siguiente ejemplo, se crea una tabla denominada `GameScores` con DynamoDB Streams habilitado. Solo se escriben en el flujo los atributos de clave del elementos modificados.

```
aws dynamodb create-table \
    --table-name GameScores \
```

```
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=TRUE,StreamViewType=KEYS_ONLY
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:45:34.140000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
```

```
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "KEYS_ONLY"
        },
        "LatestStreamLabel": "2023-05-25T18:45:34.140",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/2023-05-25T18:45:34.140",
        "DeletionProtectionEnabled": false
    }
}
```

Para obtener más información, consulte Captura de datos de cambios para DynamoDB Streams en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 9: creación de una tabla mediante la clase de tabla de acceso poco frecuente estándar de DynamoDB

En el siguiente ejemplo se crea una tabla denominada `GameScores` y asigna la clase de tabla Estándar - Acceso poco frecuente (DynamoDB Standard-IA). Esta clase de tabla está optimizada para que el almacenamiento sea el costo dominante.

```
aws dynamodb create-table \
    --table-name GameScores \
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --table-class STANDARD_INFREQUENT_ACCESS
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
```

```
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T18:33:07.581000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "TableClassSummary": {
            "TableClass": "STANDARD_INFREQUENT_ACCESS"
        },
        "DeletionProtectionEnabled": false
    }
}
```

Para obtener más información, consulte Clases de tabla en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 10: creación de una tabla con la protección contra eliminación habilitada

En el siguiente ejemplo, se crea una tabla denominada `GameScores` y habilita la protección contra eliminación.

```
aws dynamodb create-table \
    --table-name GameScores \
```

```
    --attribute-
definitions AttributeName=UserId,AttributeType=S AttributeName=GameTitle,AttributeType=S
 \
    --key-
schema AttributeName=UserId,KeyType=HASH AttributeName=GameTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --deletion-protection-enabled
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "GameTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "UserId",
                "AttributeType": "S"
            }
        ],
        "TableName": "GameScores",
        "KeySchema": [
            {
                "AttributeName": "UserId",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "GameTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2023-05-25T23:02:17.093000+00:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 10,
            "WriteCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
```

```
        "TableId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "DeletionProtectionEnabled": true
    }
}
```

Para obtener más información, consulte Uso de la protección contra eliminación en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte CreateTable en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
```

```
}



// CreateMovieTable creates a DynamoDB table with a composite primary key defined
 as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.BillingMode;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.OnDemandThroughput;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateTable {
    public static void main(String[] args) {
        final String usage = """

            Usage:
                <tableName> <key>

            Where:
                tableName - The Amazon DynamoDB table to create (for example,
 Music3).
                key - The key for the Amazon DynamoDB table (for example,
 Artist).
            """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        System.out.println("Creating an Amazon DynamoDB table " + tableName + "
 with a simple primary key: " + key);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        String result = createTable(ddb, tableName, key);
        System.out.println("New table is " + result);
        ddb.close();
    }

    public static String createTable(DynamoDbClient ddb, String tableName, String
 key) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        CreateTableRequest request = CreateTableRequest.builder()
            .attributeDefinitions(AttributeDefinition.builder()
                .attributeName(key)
                .attributeType(ScalarAttributeType.S)
                .build())
```

```
            .keySchema(KeySchemaElement.builder()
                .attributeName(key)
                .keyType(KeyType.HASH)
                .build())
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
 scales based on traffic.
            .tableName(tableName)
            .build();

        String newTable;
        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
 dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            newTable = response.tableDescription().tableName();
            return newTable;

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new CreateTableCommand({
    TableName: "EspressoDrinks",
    // For more information about data types,
    // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeDefinitions: [
      {
        AttributeName: "DrinkName",
        AttributeType: "S",
      },
    ],
    KeySchema: [
      {
        AttributeName: "DrinkName",
        KeyType: "HASH",
      },
    ],
    BillingMode: "PAY_PER_REQUEST",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  AttributeDefinitions: [
    {
      AttributeName: "CUSTOMER_ID",
      AttributeType: "N",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      AttributeType: "S",
    },
  ],
  KeySchema: [
    {
      AttributeName: "CUSTOMER_ID",
      KeyType: "HASH",
    },
    {
      AttributeName: "CUSTOMER_NAME",
      KeyType: "RANGE",
    },
  ],
```

```
  ProvisionedThroughput: {
    ReadCapacityUnits: 1,
    WriteCapacityUnits: 1,
  },
  TableName: "CUSTOMER_LIST",
  StreamSpecification: {
    StreamEnabled: false,
  },
};

// Call DynamoDB to create the table
ddb.createTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Table Created", data);
  }
});
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun createNewTable(
    tableNameVal: String,
    key: String,
): String? {
    val attDef =
        AttributeDefinition {
```

```
            attributeName = key
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef)
            keySchema = listOf(keySchemaVal)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        var tableArn: String
        val response = ddb.createTable(request)
        ddb.waitUntilTableExists {
            // suspend call
            tableName = tableNameVal
        }
        tableArn = response.tableDescription!!.tableArn.toString()
        println("Table $tableArn is ready")
        return tableArn
    }
}
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear una tabla.

```
        $tableName = "ddb_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

    public function createTable(string $tableName, array $attributes)
    {
        $keySchema = [];
        $attributeDefinitions = [];
        foreach ($attributes as $attribute) {
            if (is_a($attribute, DynamoDBAttribute::class)) {
                $keySchema[] = ['AttributeName' => $attribute->AttributeName,
'KeyType' => $attribute->KeyType];
                $attributeDefinitions[] =
                    ['AttributeName' => $attribute->AttributeName,
'AttributeType' => $attribute->AttributeType];
            }
        }

        $this->dynamoDbClient->createTable([
            'TableName' => $tableName,
            'KeySchema' => $keySchema,
            'AttributeDefinitions' => $attributeDefinitions,
            'ProvisionedThroughput' => ['ReadCapacityUnits' => 10,
'WriteCapacityUnits' => 10],
        ]);
    }
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para PHP.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: en este ejemplo se crea una tabla denominada Thread que tiene una clave principal compuesta por ForumName (hash de tipos de clave) y Subject (rango de tipos de clave). El esquema utilizado para construir la tabla se puede canalizar hacia cada cmdlet tal como se muestra o se especifica con el parámetro -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Salida:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Ejemplo 2: en este ejemplo se crea una tabla denominada Thread que tiene una clave principal compuesta por ForumName (hash de tipo clave) y Subject (rango de tipos de clave). También se define un índice secundario local. La clave del índice secundario local se establecerá automáticamente a partir de la clave hash principal de la tabla (ForumName). El esquema utilizado para construir la tabla se puede canalizar hacia cada cmdlet tal como se muestra o se especifica con el parámetro -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
```

```
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Salida:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

Ejemplo 3: en este ejemplo se muestra cómo usar una sola canalización para crear una tabla denominada Thread que tiene una clave principal compuesta por ForumName (hash de tipo clave) y Subject (rango de tipos de clave), además de un índice secundario local. Add-DDBKeySchema y Add-DDBIndexSchema crean un nuevo objeto TableSchema para usted si no se proporciona ninguno desde la canalización o el parámetro -Schema.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Salida:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
```

```
LocalSecondaryIndexes  : {LastPostIndex}
```

- Para obtener detalles de la API, consulte CreateTable en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: en este ejemplo se crea una tabla denominada Thread que tiene una clave principal compuesta por ForumName (hash de tipos de clave) y Subject (rango de tipos de clave). El esquema utilizado para construir la tabla se puede canalizar hacia cada cmdlet tal como se muestra o se especifica con el parámetro -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyType RANGE -KeyDataType "S"
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Salida:

```
AttributeDefinitions   : {ForumName, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {}
```

Ejemplo 2: en este ejemplo se crea una tabla denominada Thread que tiene una clave principal compuesta por ForumName (hash de tipo clave) y Subject (rango de tipos de clave). También se define un índice secundario local. La clave del índice secundario local se establecerá automáticamente a partir de la clave hash principal de la tabla (ForumName). El esquema utilizado para construir la tabla se puede canalizar hacia cada cmdlet tal como se muestra o se especifica con el parámetro -Schema.

```
$schema = New-DDBTableSchema
$schema | Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S"
$schema | Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S"
$schema | Add-DDBIndexSchema -IndexName "LastPostIndex" -RangeKeyName
 "LastPostDateTime" -RangeKeyDataType "S" -ProjectionType "keys_only"
```

```
$schema | New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Salida:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

Ejemplo 3: en este ejemplo se muestra cómo usar una sola canalización para crear una tabla denominada Thread que tiene una clave principal compuesta por ForumName (hash de tipo clave) y Subject (rango de tipos de clave), además de un índice secundario local. Add-DDBKeySchema y Add-DDBIndexSchema crean un nuevo objeto TableSchema para usted si no se proporciona ninguno desde la canalización o el parámetro -Schema.

```
New-DDBTableSchema |
  Add-DDBKeySchema -KeyName "ForumName" -KeyDataType "S" |
  Add-DDBKeySchema -KeyName "Subject" -KeyDataType "S" |
  Add-DDBIndexSchema -IndexName "LastPostIndex" `
                     -RangeKeyName "LastPostDateTime" `
                     -RangeKeyDataType "S" `
                     -ProjectionType "keys_only" |
  New-DDBTable -TableName "Thread" -ReadCapacity 10 -WriteCapacity 5
```

Salida:

```
AttributeDefinitions   : {ForumName, LastPostDateTime, Subject}
TableName              : Thread
KeySchema              : {ForumName, Subject}
TableStatus            : CREATING
CreationDateTime       : 10/28/2013 4:39:49 PM
ProvisionedThroughput  : Amazon.DynamoDBv2.Model.ProvisionedThroughputDescription
TableSizeBytes         : 0
ItemCount              : 0
LocalSecondaryIndexes  : {LastPostIndex}
```

- Para obtener detalles de la API, consulte CreateTable en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Cree una tabla para almacenar datos de películas.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
```

```
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def create_table(self, table_name):
        """
        Creates an Amazon DynamoDB table that can be used to store movie data.
        The table uses the release year of the movie as the partition key and the
        title as the sort key.

        :param table_name: The name of the table to create.
        :return: The newly created table.
        """
        try:
            self.table = self.dyn_resource.create_table(
                TableName=table_name,
                KeySchema=[
                    {"AttributeName": "year", "KeyType": "HASH"},  # Partition
key
                    {"AttributeName": "title", "KeyType": "RANGE"},  # Sort key
                ],
                AttributeDefinitions=[
                    {"AttributeName": "year", "AttributeType": "N"},
                    {"AttributeName": "title", "AttributeType": "S"},
                ],
                BillingMode='PAY_PER_REQUEST',
            )
            self.table.wait_until_exists()
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.table
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Creates an Amazon DynamoDB table that can be used to store movie data.
  # The table uses the release year of the movie as the partition key and the
  # title as the sort key.
  #
  # @param table_name [String] The name of the table to create.
  # @return [Aws::DynamoDB::Table] The newly created table.
  def create_table(table_name)
    @table = @dynamo_resource.create_table(
      table_name: table_name,
      key_schema: [
        { attribute_name: 'year', key_type: 'HASH' }, # Partition key
        { attribute_name: 'title', key_type: 'RANGE' } # Sort key
      ],
      attribute_definitions: [
        { attribute_name: 'year', attribute_type: 'N' },
```

```
        { attribute_name: 'title', attribute_type: 'S' }
      ],
      billing_mode: 'PAY_PER_REQUEST'
    )
    @dynamo_resource.client.wait_until(:table_exists, table_name: table_name)
    @table
  rescue Aws::DynamoDB::Errors::ServiceError => e
    @logger.error("Failed create table #{table_name}:\n#{e.code}: #{e.message}")
    raise
  end
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
pub async fn create_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<CreateTableOutput, Error> {
    let a_name: String = key.into();
    let table_name: String = table.into();

    let ad = AttributeDefinition::builder()
        .attribute_name(&a_name)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .map_err(Error::BuildError)?;

    let ks = KeySchemaElement::builder()
        .attribute_name(&a_name)
```

```
        .key_type(KeyType::Hash)
        .build()
        .map_err(Error::BuildError)?;

    let create_table_response = client
        .create_table()
        .table_name(table_name)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .send()
        .await;

    match create_table_response {
        Ok(out) => {
            println!("Added table {} with key {}", table, key);
            Ok(out)
        }
        Err(e) => {
            eprintln!("Got an error creating table:");
            eprintln!("{}", e);
            Err(Error::unhandled(e))
        }
    }
}
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    TRY.
```

```
        DATA(lt_keyschema) = VALUE /aws1/cl_dynkeyschemaelement=>tt_keyschema(
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'year'
                                              iv_keytype = 'HASH' ) )
          ( NEW /aws1/cl_dynkeyschemaelement( iv_attributename = 'title'
                                              iv_keytype = 'RANGE' ) ) ).
        DATA(lt_attributedefinitions) = VALUE /aws1/
cl_dynattributedefn=>tt_attributedefinitions(
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'year'
                                           iv_attributetype = 'N' ) )
          ( NEW /aws1/cl_dynattributedefn( iv_attributename = 'title'
                                           iv_attributetype = 'S' ) ) ).

        " Adjust read/write capacities as desired.
        DATA(lo_dynprovthroughput)  = NEW /aws1/cl_dynprovthroughput(
          iv_readcapacityunits = 5
          iv_writecapacityunits = 5 ).
        oo_result = lo_dyn->createtable(
          it_keyschema = lt_keyschema
          iv_tablename = iv_table_name
          it_attributedefinitions = lt_attributedefinitions
          io_provisionedthroughput = lo_dynprovthroughput ).
        " Table creation can take some time. Wait till table exists before
 returning.
        lo_dyn->get_waiter( )->tableexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'DynamoDB Table' && iv_table_name && 'created.' TYPE 'I'.
        " This exception can happen if the table already exists.
      CATCH /aws1/cx_dynresourceinuseex INTO DATA(lo_resourceinuseex).
        DATA(lv_error) = |"{ lo_resourceinuseex->av_err_code }" -
{ lo_resourceinuseex->av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- Para obtener información sobre la API, consulte CreateTable en la Referencia de la API del AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB


    ///
    /// Create a movie table in the Amazon DynamoDB data store.
    ///
    private func createTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = CreateTableInput(
                attributeDefinitions: [
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
 "year", attributeType: .n),
                    DynamoDBClientTypes.AttributeDefinition(attributeName:
 "title", attributeType: .s)
                ],
                billingMode: DynamoDBClientTypes.BillingMode.payPerRequest,
                keySchema: [
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "year",
 keyType: .hash),
                    DynamoDBClientTypes.KeySchemaElement(attributeName: "title",
 keyType: .range)
                ],
                tableName: self.tableName
            )
            let output = try await client.createTable(input: input)
            if output.tableDescription == nil {
                throw MoviesError.TableNotFound
            }
```

```
        } catch {
            print("ERROR: createTable:", dump(error))
            throw error
        }
    }
```

- Para obtener detalles sobre la API, consulte CreateTable en la Referencia de la API del SDK de AWS para Swift.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de **`DeleteItem`** con un SDK de AWS o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `DeleteItem`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Conceptos básicos
- Uso de operaciones condicionales

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Deletes a single item from a DynamoDB table.
/// </summary>
/// <param name="tableName">The name of the table from which the item
```

```
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N =
movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

- Para obtener información sobre la API, consulte DeleteItem en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_delete_item
#
# This function deletes an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_item() {
  local table_name keys response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_delete_item"
    echo "Delete an item from a DynamoDB table."
    echo " -n table_name  -- The name of the table."
```

```
    echo " -k keys  -- Path to json file containing the keys that identify the
item to delete."
    echo ""
 }
 while getopts "n:k:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      k) keys="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
 fi

 if [[ -z "$keys" ]]; then
    errecho "ERROR: You must provide a keys json file path the -k parameter."
    usage
    return 1
 fi

 iecho "Parameters:\n"
 iecho "    table_name:   $table_name"
 iecho "    keys:   $keys"
 iecho ""

 response=$(aws dynamodb delete-item \
    --table-name "$table_name" \
    --key file://"$keys")

 local error_code=${?}
```

```
  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-item operation failed.$response"
    return 1
  fi

  return 0

}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
```

```
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información sobre la API, consulte DeleteItem en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Delete an item from an Amazon DynamoDB table.
/*!
  \sa deleteItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::deleteItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteItemRequest request;

    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DeleteItemOutcome &outcome =
 dynamoClient.DeleteItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item \"" << partitionValue << "\" deleted!" << std::endl;
    }
    else {
        std::cerr << "Failed to delete item: " << outcome.GetError().GetMessage()
                  << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Código que espera a que se active la tabla.

```
//! Query a newly created DynamoDB table until it is active.
/*!
```

```
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obtener información sobre la API, consulte DeleteItem en la referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Ejemplo 1: Eliminación de un elemento

En el siguiente ejemplo `delete-item`, se elimina un elemento de la tabla `MusicCollection` y se solicitan detalles sobre el elemento que se ha eliminado y la capacidad utilizada por la solicitud.

```
aws dynamodb delete-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-values ALL_OLD \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Contenido de `key.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Scared of My Shadow"}
}
```

Salida:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Blue Sky Blues"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Scared of My Shadow"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 2.0
    },
```

```
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Para obtener más información, consulte Escritura de un elemento en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 2: Eliminación de un elemento de forma condicional

En el siguiente ejemplo, se elimina un elemento de la tabla `ProductCatalog` solo si `ProductCategory` es `Sporting Goods` o `Gardening Supplies` y su precio está comprendido entre 500 y 600. Devuelve detalles sobre el elemento que se ha eliminado.

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{"Id":{"N":"456"}}' \
    --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (#P
 between :lo and :hi)" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

Contenido de `names.json`:

```
{
    "#P": "Price"
}
```

Contenido de `values.json`:

```
{
    ":cat1": {"S": "Sporting Goods"},
```

```
    ":cat2": {"S": "Gardening Supplies"},
    ":lo": {"N": "500"},
    ":hi": {"N": "600"}
}
```

Salida:

```
{
    "Attributes": {
        "Id": {
            "N": "456"
        },
        "Price": {
            "N": "550"
        },
        "ProductCategory": {
            "S": "Sporting Goods"
        }
    }
}
```

Para obtener más información, consulte Escritura de un elemento en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte DeleteItem en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
 "errors"
```

```
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(ctx context.Context, movie Movie) error {
 _, err := basics.DynamoDbClient.DeleteItem(ctx, &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

Defina una estructura Película para utilizar en este ejemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
```

```
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obtener información sobre la API, consulte DeleteItem en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyval>

                Where:
                    tableName - The Amazon DynamoDB table to delete the item from
 (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
 Artist).\s
```

```
                    keyval - The key value that represents the item to delete
 (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        System.out.format("Deleting item \"%s\" from %s\n", keyVal, tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBItem(ddb, tableName, key, keyVal);
        ddb.close();
    }

    public static void deleteDynamoDBItem(DynamoDbClient ddb, String tableName,
 String key, String keyVal) {
        HashMap<String, AttributeValue> keyToGet = new HashMap<>();
        keyToGet.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        DeleteItemRequest deleteReq = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(keyToGet)
                .build();

        try {
            ddb.deleteItem(deleteReq);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obtener información sobre la API, consulte DeleteItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener información sobre la API, consulte DeleteCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, DeleteCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte DeleteItem en la referencia de la API de AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Eliminar un elemento de una tabla.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "VALUE" },
  },
};

// Call DynamoDB to delete the item from the table
ddb.deleteItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Eliminar un elemento de una tabla con el cliente de documentos de DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  Key: {
    HASH_KEY: VALUE,
  },
  TableName: "TABLE",
};

docClient.delete(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte DeleteItem en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun deleteDynamoDBItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)
```

```
    val request =
        DeleteItemRequest {
            tableName = tableNameVal
            key = keyToGet
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteItem(request)
        println("Item with key matching $keyVal was deleted")
    }
}
```

- Para obtener información sobre la API, consulte DeleteItem en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        $key = [
            'Item' => [
                'title' => [
                    'S' => $movieName,
                ],
                'year' => [
                    'N' => $movieYear,
                ],
            ]
        ];

        $service->deleteItemByKey($tableName, $key);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";
```

```
    public function deleteItemByKey(string $tableName, array $key)
    {
        $this->dynamoDbClient->deleteItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- Para obtener información sobre la API, consulte DeleteItem en la referencia de la API de AWS SDK para PHP.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: elimina el elemento DynamoDB que coincide con la clave proporcionada.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- Para obtener detalles de la API, consulte DeleteItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: elimina el elemento DynamoDB que coincide con la clave proporcionada.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem
Remove-DDBItem -TableName 'Music' -Key $key -Confirm:$false
```

- Para obtener detalles de la API, consulte DeleteItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def delete_movie(self, title, year):
        """
        Deletes a movie from the table.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        """
        try:
            self.table.delete_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't delete movie %s. Here's why: %s: %s",
                title,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

Puede especificar una condición para que un elemento se elimine solo cuando cumpla ciertos criterios.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def delete_underrated_movie(self, title, year, rating):
        """
        Deletes a movie only if it is rated below a specified value. By using a
        condition expression in a delete operation, you can specify that an item
 is
        deleted only when it meets certain criteria.

        :param title: The title of the movie to delete.
        :param year: The release year of the movie to delete.
        :param rating: The rating threshold to check before deleting the movie.
        """
        try:
            self.table.delete_item(
                Key={"year": year, "title": title},
```

```
                ConditionExpression="info.rating <= :val",
                ExpressionAttributeValues={":val": Decimal(str(rating))},
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
 "ConditionalCheckFailedException":
                logger.warning(
                    "Didn't delete %s because its rating is greater than %s.",
                    title,
                    rating,
                )
            else:
                logger.error(
                    "Couldn't delete movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
```

- Para obtener información sobre la API, consulte DeleteItem en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
```

```
    @table = @dynamo_resource.table(table_name)
  end

  # Deletes a movie from the table.
  #
  # @param title [String] The title of the movie to delete.
  # @param year [Integer] The release year of the movie to delete.
  def delete_item(title, year)
    @table.delete_item(key: { 'year' => year, 'title' => title })
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete movie #{title}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Para obtener información sobre la API, consulte DeleteItem en la referencia de la API de AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
pub async fn delete_item(
    client: &Client,
    table: &str,
    key: &str,
    value: &str,
) -> Result<DeleteItemOutput, Error> {
    match client
        .delete_item()
        .table_name(table)
        .key(key, AttributeValue::S(value.into()))
        .send()
        .await
```

```
    {
        Ok(out) => {
            println!("Deleted item from table");
            Ok(out)
        }
        Err(e) => Err(Error::unhandled(e)),
    }
}
```

- Para obtener información sobre la API, consulte DeleteItem en la referencia de la API de AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    TRY.
        DATA(lo_resp) = lo_dyn->deleteitem(
          iv_tablename                = iv_table_name
          it_key                      = it_key_input ).
        MESSAGE 'Deleted one item.' TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Para obtener información sobre las API, consulte DeleteItem en la Referencia de la API del SDK AWS para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB


    /// Delete a movie, given its title and release year.
    ///
    /// - Parameters:
    ///   - title: The movie's title.
    ///   - year: The movie's release year.
    ///
    func delete(title: String, year: Int) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteItemInput(
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            _ = try await client.deleteItem(input: input)
        } catch {
            print("ERROR: delete:", dump(error))
            throw error
        }
    }
```

- Para obtener más detalles sobre la API, consulte DeleteItem en la referencia de la API del SDK de AWS para Swift.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de **`DeleteTable`** con un SDK de AWS o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `DeleteTable`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Conceptos básicos
- Aceleración de lecturas con DAX
- Creación y administración de tablas globales de MRSC
- Administración de índices secundarios globales
- Trabajo con tablas globales y coherencia final de replicación de varias regiones (MREC)

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
```

```
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table
{tableName}. {ex.Message}");
            return false;
        }
    }
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_delete_table
#
# This function deletes a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table to delete.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_delete_table() {
  local table_name response
  local option OPTARG # Required to use getopts command in a function.

  # bashsupport disable=BP5008
  function usage() {
    echo "function dynamodb_delete_table"
    echo "Deletes an Amazon DynamoDB table."
    echo " -n table_name  -- The name of the table to delete."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
```

```
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho ""

  response=$(aws dynamodb delete-table \
    --table-name "$table_name")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports delete-table operation failed.$response"
    return 1
  fi

  return 0
}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
```

```
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

# bash_error_handling_function_1
# bash_error_handling_function_2
# bash_error_handling_function_3
# bash_error_handling_function_4
# bash_error_handling_function_5
# bash_error_handling_function_6
# bash_error_handling_function_7
# bash_error_handling_function_8
# bash_error_handling_function_9
# bash_error_handling_function_10
# bash_error_handling_function_11
# bash_error_handling_function_12
# bash_error_handling_function_13
# bash_error_handling_function_14
# bash_error_handling_function_15
# bash_error_handling_function_16
# bash_error_handling_function_17
# bash_error_handling_function_18
# bash_error_handling_function_19
# bash_error_handling_function_20
#
```

```
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información sobre la API, consulte DeleteTable en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Delete an Amazon DynamoDB table.
/*!
  \sa deleteTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteTable(const Aws::String &tableName,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
```

```
                << result.GetResult().GetTableDescription().GetTableName()
                << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                << std::endl;
    }

    return result.IsSuccess();
}
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Eliminación de una tabla

En el siguiente ejemplo de `delete-table` se elimina la tabla `MusicCollection`.

```
aws dynamodb delete-table \
    --table-name MusicCollection
```

Salida:

```
{
    "TableDescription": {
        "TableStatus": "DELETING",
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableName": "MusicCollection",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        }
    }
}
```

Para obtener más información, consulte Eliminación de una tabla en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte DeleteTable en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteTable deletes the DynamoDB table and all of its data.
```

```
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
 return err
}
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */

public class DeleteTable {
    public static void main(String[] args) {
```

```
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to delete (for example,
Music3).

                **Warning** This program will delete the table that you specify!
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        System.out.format("Deleting the Amazon DynamoDB table %s...\n",
tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
{
        DeleteTableRequest request = DeleteTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }
```

```
}
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to delete the specified table
ddb.deleteTable(params, function (err, data) {
  if (err && err.code === "ResourceNotFoundException") {
    console.log("Error: Table not found");
  } else if (err && err.code === "ResourceInUseException") {
    console.log("Error: Table in use");
  } else {
    console.log("Success", data);
  }
});
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun deleteDynamoDBTable(tableNameVal: String) {
```

```
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    public function deleteTable(string $TableName)
    {
        $this->customWaiter(function () use ($TableName) {
            return $this->dynamoDbClient->deleteTable([
                'TableName' => $TableName,
            ]);
        });
    }
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para PHP.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: elimina la tabla especificada. Se le solicitará una confirmación antes de continuar con la operación.

```
Remove-DDBTable -TableName "myTable"
```

Ejemplo 2: elimina la tabla especificada. No se le solicitará una confirmación antes de continuar con la operación.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Para obtener detalles de la API, consulte DeleteTable en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: elimina la tabla especificada. Se le solicitará una confirmación antes de continuar con la operación.

```
Remove-DDBTable -TableName "myTable"
```

Ejemplo 2: elimina la tabla especificada. No se le solicitará una confirmación antes de continuar con la operación.

```
Remove-DDBTable -TableName "myTable" -Force
```

- Para obtener detalles de la API, consulte DeleteTable en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def delete_table(self):
        """
        Deletes the table.
        """
        try:
            self.table.delete()
            self.table = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table. Here's why: %s: %s",
```

```
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Deletes the table.
  def delete_table
    @table.delete
    @table = nil
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't delete table. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
```

```
  end
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
pub async fn delete_table(client: &Client, table: &str) ->
 Result<DeleteTableOutput, Error> {
    let resp = client.delete_table().table_name(table).send().await;

    match resp {
        Ok(out) => {
            println!("Deleted table");
            Ok(out)
        }
        Err(e) => Err(Error::Unhandled(e.into())),
    }
}
```

- Para obtener información sobre la API, consulte DeleteTable en la referencia de la API de AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    TRY.
        lo_dyn->deletetable( iv_tablename = iv_table_name ).
        " Wait till the table is actually deleted.
        lo_dyn->get_waiter( )->tablenotexists(
          iv_max_wait_time = 200
          iv_tablename     = iv_table_name ).
        MESSAGE 'Table ' && iv_table_name && ' deleted.' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table ' && iv_table_name && ' does not exist' TYPE 'E'.
      CATCH /aws1/cx_dynresourceinuseex.
        MESSAGE 'The table cannot be deleted since it is in use' TYPE 'E'.
    ENDTRY.
```

- Para obtener información sobre la API, consulte DeleteTable en la Referencia de la API del AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB
```

```
    ///
    /// Deletes the table from Amazon DynamoDB.
    ///
    func deleteTable() async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = DeleteTableInput(
                tableName: self.tableName
            )
            _ = try await client.deleteTable(input: input)
        } catch {
            print("ERROR: deleteTable:", dump(error))
            throw error
        }
    }
```

- Para obtener información acerca de la API, consulte DeleteTable en la Referencia de la API del SDK de AWS para Swift.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de **`DescribeTable`** con un SDK de AWS o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `DescribeTable`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Conceptos básicos
- Creación y administración de tablas globales que muestren MREC
- Creación y administración de tablas globales de MRSC
- Trabajo con tablas globales y coherencia final de replicación de varias regiones (MREC)
- Trabajo con Streams y tiempo de vida

- Trabajo con el cifrado de tablas

.NET

SDK para .NET

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
private static async Task GetTableInformation()
{
    Console.WriteLine("\n*** Retrieving table information ***");

    var response = await Client.DescribeTableAsync(new DescribeTableRequest
    {
        TableName = ExampleTableName
    });

    var table = response.Table;
    Console.WriteLine($"Name: {table.TableName}");
    Console.WriteLine($"# of items: {table.ItemCount}");

}
```

- Para obtener información sobre la API, consulte DescribeTable en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_describe_table
#
# This function returns the status of a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#
#  Response:
#       - TableStatus:
#     And:
#       0 - Table is active.
#       1 - If it fails.
###############################################################################
function dynamodb_describe_table {
  local table_name
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_describe_table"
    echo "Describe the status of a DynamoDB table."
    echo "  -n table_name  -- The name of the table."
    echo ""
  }

  # Retrieve the calling parameters.
  while getopts "n:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
```

```
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  local table_status
    table_status=$(
      aws dynamodb describe-table \
        --table-name "$table_name" \
        --output text \
        --query 'Table.TableStatus'
    )

   local error_code=${?}

    if [[ $error_code -ne 0 ]]; then
      aws_cli_error_log "$error_code"
      errecho "ERROR: AWS reports describe-table operation failed.$table_status"
      return 1
    fi

  echo "$table_status"

  return 0
}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
```

```
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
########################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información sobre la API, consulte DescribeTable en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Describe an Amazon DynamoDB table.
/*!
  \sa describeTable()
  \param tableName: The DynamoDB table name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::describeTable(const Aws::String &tableName,
                                     const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    const Aws::DynamoDB::Model::DescribeTableOutcome &outcome =
 dynamoClient.DescribeTable(
            request);

    if (outcome.IsSuccess()) {
        const Aws::DynamoDB::Model::TableDescription &td =
 outcome.GetResult().GetTable();
        std::cout << "Table name  : " << td.GetTableName() << std::endl;
        std::cout << "Table ARN   : " << td.GetTableArn() << std::endl;
        std::cout << "Status      : "
                  <<
 Aws::DynamoDB::Model::TableStatusMapper::GetNameForTableStatus(
                          td.GetTableStatus()) << std::endl;
        std::cout << "Item count  : " << td.GetItemCount() << std::endl;
        std::cout << "Size (bytes): " << td.GetTableSizeBytes() << std::endl;
```

```
        const Aws::DynamoDB::Model::ProvisionedThroughputDescription &ptd =
 td.GetProvisionedThroughput();
        std::cout << "Throughput" << std::endl;
        std::cout << "  Read Capacity : " << ptd.GetReadCapacityUnits() <<
 std::endl;
        std::cout << "  Write Capacity: " << ptd.GetWriteCapacityUnits() <<
 std::endl;

        const Aws::Vector<Aws::DynamoDB::Model::AttributeDefinition> &ad =
 td.GetAttributeDefinitions();
        std::cout << "Attributes" << std::endl;
        for (const auto &a: ad)
            std::cout << "  " << a.GetAttributeName() << " (" <<

 Aws::DynamoDB::Model::ScalarAttributeTypeMapper::GetNameForScalarAttributeType(
                                a.GetAttributeType()) <<
                     ")" << std::endl;
    }
    else {
        std::cerr << "Failed to describe table: " <<
 outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
}
```

- Para obtener información sobre la API, consulte DescribeTable en la Referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Descripción de una tabla

En el siguiente ejemplo `describe-table`, se describe la tabla `MusicCollection`.

```
aws dynamodb describe-table \
    --table-name MusicCollection
```

Salida:

```
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "MusicCollection",
        "TableStatus": "ACTIVE",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Artist"
            },
            {
                "KeyType": "RANGE",
                "AttributeName": "SongTitle"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1421866952.062
    }
}
```

Para obtener más información, consulte Descripción de una tabla en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte DescribeTable en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// TableExists determines whether a DynamoDB table exists.
func (basics TableBasics) TableExists(ctx context.Context) (bool, error) {
 exists := true
 _, err := basics.DynamoDbClient.DescribeTable(
  ctx, &dynamodb.DescribeTableInput{TableName: aws.String(basics.TableName)},
 )
 if err != nil {
```

```
  var notFoundEx *types.ResourceNotFoundException
  if errors.As(err, &notFoundEx) {
   log.Printf("Table %v does not exist.\n", basics.TableName)
   err = nil
  } else {
   log.Printf("Couldn't determine existence of table %v. Here's why: %v\n",
 basics.TableName, err)
  }
  exists = false
 }
 return exists, err
}
```

- Para obtener información sobre la API, consulte DescribeTable en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import
 software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughputDescription;
import software.amazon.awssdk.services.dynamodb.model.TableDescription;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
```

```
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DescribeTable {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to get information
 about (for example, Music3).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        System.out.format("Getting description for %s\n\n", tableName);
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        describeDymamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void describeDymamoDBTable(DynamoDbClient ddb, String
 tableName) {
        DescribeTableRequest request = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

        try {
            TableDescription tableInfo = ddb.describeTable(request).table();
            if (tableInfo != null) {
                System.out.format("Table name  : %s\n", tableInfo.tableName());
```

```
                System.out.format("Table ARN   : %s\n", tableInfo.tableArn());
                System.out.format("Status      : %s\n", tableInfo.tableStatus());
                System.out.format("Item count  : %d\n", tableInfo.itemCount());
                System.out.format("Size (bytes): %d\n",
 tableInfo.tableSizeBytes());

                ProvisionedThroughputDescription throughputInfo =
 tableInfo.provisionedThroughput();
                System.out.println("Throughput");
                System.out.format("  Read Capacity : %d\n",
 throughputInfo.readCapacityUnits());
                System.out.format("  Write Capacity: %d\n",
 throughputInfo.writeCapacityUnits());

                List<AttributeDefinition> attributes =
 tableInfo.attributeDefinitions();
                System.out.println("Attributes");
                for (AttributeDefinition a : attributes) {
                    System.out.format("  %s (%s)\n", a.attributeName(),
 a.attributeType());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("\nDone!");
    }
}
```

- Para obtener información sobre la API, consulte DescribeTable en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import { DescribeTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte DescribeTable en la referencia de la API de AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
```

```
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: process.argv[2],
};

// Call DynamoDB to retrieve the selected table descriptions
ddb.describeTable(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Table.KeySchema);
  }
});
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte DescribeTable en la referencia de la API de AWS SDK para JavaScript.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: devuelve información detallada de la tabla especificada.

```
Get-DDBTable -TableName "myTable"
```

- Para obtener detalles de la API, consulte DescribeTable en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: devuelve información detallada de la tabla especificada.

```
Get-DDBTable -TableName "myTable"
```

- Para obtener detalles de la API, consulte DescribeTable en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
```

```
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def exists(self, table_name):
        """
        Determines whether a table exists. As a side effect, stores the table in
        a member variable.

        :param table_name: The name of the table to check.
        :return: True when the table exists; otherwise, False.
        """
        try:
            table = self.dyn_resource.Table(table_name)
            table.load()
            exists = True
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                exists = False
            else:
                logger.error(
                    "Couldn't check for existence of %s. Here's why: %s: %s",
                    table_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            self.table = table
        return exists
```

- Para obtener información sobre la API, consulte DescribeTable en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Determines whether a table exists. As a side effect, stores the table in
  # a member variable.
  #
  # @param table_name [String] The name of the table to check.
  # @return [Boolean] True when the table exists; otherwise, False.
  def exists?(table_name)
    @dynamo_resource.client.describe_table(table_name: table_name)
    @logger.debug("Table #{table_name} exists")
  rescue Aws::DynamoDB::Errors::ResourceNotFoundException
    @logger.debug("Table #{table_name} doesn't exist")
    false
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't check for existence of #{table_name}:\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Para obtener información sobre la API, consulte DescribeTable en la referencia de la API de AWS SDK para Ruby.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
TRY.
    oo_result = lo_dyn->describetable( iv_tablename = iv_table_name ).
    DATA(lv_tablename) = oo_result->get_table( )->ask_tablename( ).
    DATA(lv_tablearn) = oo_result->get_table( )->ask_tablearn( ).
    DATA(lv_tablestatus) = oo_result->get_table( )->ask_tablestatus( ).
    DATA(lv_itemcount) = oo_result->get_table( )->ask_itemcount( ).
    MESSAGE 'The table name is ' && lv_tablename
        && '. The table ARN is ' && lv_tablearn
        && '. The tablestatus is ' && lv_tablestatus
        && '. Item count is ' && lv_itemcount TYPE 'I'.
  CATCH /aws1/cx_dynresourcenotfoundex.
    MESSAGE 'The table ' && lv_tablename && ' does not exist' TYPE 'E'.
ENDTRY.
```

- Para obtener información acerca de la API, consulte DescribeTable en la Referencia de API del SDK AWS para SAP ABAP.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de **`DescribeTimeToLive`** con un SDK de AWS o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `DescribeTimeToLive`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- Trabajo con Streams y tiempo de vida

CLI

AWS CLI

Visualización de la configuración de tiempo de vida de una tabla

En el siguiente ejemplo de `describe-time-to-live`, se muestra la configuración de tiempo de vida de la tabla `MusicCollection`.

```
aws dynamodb describe-time-to-live \
    --table-name MusicCollection
```

Salida:

```
{
    "TimeToLiveDescription": {
        "TimeToLiveStatus": "ENABLED",
        "AttributeName": "ttl"
    }
}
```

Para obtener más información, consulte Tiempo de vida en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte DescribeTimeToLive en la Referencia de comandos de la AWS CLI.

Java

SDK para Java 2.x

Se describe la configuración de TTL en una tabla de DynamoDB existente mediante AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DescribeTimeToLiveRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTimeToLiveResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.logging.Level;
import java.util.logging.Logger;

    public DescribeTimeToLiveResponse describeTTL(final String tableName, final
 Region region) {
        final DescribeTimeToLiveRequest request =
            DescribeTimeToLiveRequest.builder().tableName(tableName).build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            return ddb.describeTimeToLive(request);
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- Para obtener detalles sobre la API, consulte DescribeTimeToLive en la Referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Se describe la configuración de TTL en una tabla de DynamoDB existente mediante AWS SDK para JavaScript.

```
import { DynamoDBClient, DescribeTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";

export const describeTTL = async (tableName, region) => {
```

```
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    try {
        const ttlDescription = await client.send(new
 DescribeTimeToLiveCommand({ TableName: tableName }));

        if (ttlDescription.TimeToLiveDescription.TimeToLiveStatus === 'ENABLED')
 {
            console.log("TTL is enabled for table %s.", tableName);
        } else {
            console.log("TTL is not enabled for table %s.", tableName);
        }

        return ttlDescription;
    } catch (e) {
        console.error(`Error describing table: ${e}`);
        throw e;
    }
}

// Example usage (commented out for testing)
// describeTTL('your-table-name', 'us-east-1');
```

- Para obtener detalles sobre la API, consulte DescribeTimeToLive en la Referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Se describe la configuración de TTL en una tabla de DynamoDB existente mediante AWS SDK for Python (Boto3).

```
import boto3


def describe_ttl(table_name, region):
    """
    Describes TTL on an existing table, as well as a region.
```

```
    :param table_name: String representing the name of the table
    :param region: AWS Region of the table - example `us-east-1`
    :return: Time to live description.
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        ttl_description = dynamodb.describe_time_to_live(TableName=table_name)
        print(
            f"TimeToLive for table {table_name} is status
 {ttl_description['TimeToLiveDescription']['TimeToLiveStatus']}"
        )

        return ttl_description
    except Exception as e:
        print(f"Error describing table: {e}")
        raise


# Enter your own table name and AWS region
describe_ttl("your-table-name", "us-east-1")
```

- Para obtener información sobre la API, consulte DescribeTimeToLive en la Referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de **`ExecuteStatement`** con un SDK de AWS

Los siguientes ejemplos de código muestran cómo utilizar `ExecuteStatement`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Eliminación de datos mediante DELETE de PartiQL
- Inserción de datos mediante INSERT de PartiQL
- Consultar una tabla con PartiQL
- Consulta de datos mediante SELECT de PartiQL

- Actualización de datos mediante UPDATE de PartiQL

.NET

SDK para .NET

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Utilizar una instrucción INSERT para agregar un elemento.

```
        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Utilizar una instrucción SELECT para obtener un elemento.

```
        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

Utilizar una instrucción SELECT para obtener una lista de elementos.

```
        /// <summary>
        /// Retrieve multiple movies by year using a SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="year">The year the movies were released.</param>
```

```
        /// <returns></returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { N = year.ToString() },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

Utilizar una instrucción UPDATE para actualizar un elemento.

```
        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
```

```
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

Utilizar una instrucción DELETE para eliminar una sola película.

```
        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
```

```
        }
```

- Para obtener información sobre la API, consulte ExecuteStatement en la referencia de la API de AWS SDK para .NET.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Utilizar una instrucción INSERT para agregar un elemento.

```
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // 2. Add a new movie using an "Insert" statement. (ExecuteStatement)
    Aws::String title;
    float rating;
    int year;
    Aws::String plot;
    {
        title = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        year = askQuestionForInt("What year was it released? ");
        rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
it? ",
                                          1, 10);
        plot = askQuestion("Summarize the plot for me: ");

        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        request.SetStatement(sqlStream.str());
```

```
       // Create the parameter attributes.
       Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

       Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

       std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
               ALLOCATION_TAG.c_str());
       ratingAttribute->SetN(rating);
       infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

       std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
               ALLOCATION_TAG.c_str());
       plotAttribute->SetS(plot);
       infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
       attributes.push_back(infoMapAttribute);
       request.SetParameters(attributes);

       Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
               request);

       if (!outcome.IsSuccess()) {
           std::cerr << "Failed to add a movie: " <<
outcome.GetError().GetMessage()
                     << std::endl;
           return false;
       }
   }
```

Utilizar una instrucción SELECT para obtener un elemento.

```
   //  3. Get the data for the movie using a "Select" statement.
(ExecuteStatement)
   {
       Aws::DynamoDB::Model::ExecuteStatementRequest request;
       std::stringstream sqlStream;
       sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
```

```
                << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to retrieve movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
        else {
            // Print the retrieved movie information.
            const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();

            if (items.size() == 1) {
                printMovieInfo(items[0]);
            }
            else {
                std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                          << " There should be only one movie." << std::endl;
            }
        }
    }
```

Utilizar una instrucción UPDATE para actualizar un elemento.

```
    //  4. Update the data for the movie using an "Update" statement.
(ExecuteStatement)
    {
```

```
        rating = askQuestionForFloatRange(
                Aws::String("\nLet's update your movie.\nYou rated it  ") +
                std::to_string(rating)
                + ", what new rating would you give it? ", 1, 10);

        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(rating));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update a movie: "
                      << outcome.GetError().GetMessage();
            return false;
        }
    }
```

Utilizar una instrucción DELETE para eliminar un elemento.

```
    // 6. Delete the movie using a "Delete" statement. (ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());
```

```
        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movie: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }
```

- Para obtener información sobre la API, consulte ExecuteStatement en la referencia de la API de AWS SDK para C++.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Defina una estructura de receptor de funciones para el ejemplo.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
```

```
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

Utilizar una instrucción INSERT para agregar un elemento.

```
// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't insert an item with PartiQL. Here's why: %v\n", err)
 }
 return err
}
```

Utilizar una instrucción SELECT para obtener un elemento.

```
// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB
 table by
```

```
// title and year.
func (runner PartiQLRunner) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 var movie Movie
 params, err := attributevalue.MarshalList([]interface{}{title, year})
 if err != nil {
  panic(err)
 }
 response, err := runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Items[0], &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

Utilizar una instrucción SELECT para obtener una lista de elementos y proyectar los resultados.

```
// GetAllMovies runs a PartiQL SELECT statement to get all movies from the
 DynamoDB table.
// pageSize is not typically required and is used to show how to paginate the
 results.
// The results are projected to return only the title and rating of each movie.
func (runner PartiQLRunner) GetAllMovies(ctx context.Context, pageSize int32)
 ([]map[string]interface{}, error) {
 var output []map[string]interface{}
 var response *dynamodb.ExecuteStatementOutput
 var err error
 var nextToken *string
```

```
 for moreData := true; moreData; {
  response, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
   Statement: aws.String(
    fmt.Sprintf("SELECT title, info.rating FROM \"%v\"", runner.TableName)),
   Limit:     aws.Int32(pageSize),
   NextToken: nextToken,
  })
  if err != nil {
   log.Printf("Couldn't get movies. Here's why: %v\n", err)
   moreData = false
  } else {
   var pageOutput []map[string]interface{}
   err = attributevalue.UnmarshalListOfMaps(response.Items, &pageOutput)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    log.Printf("Got a page of length %v.\n", len(response.Items))
    output = append(output, pageOutput...)
   }
   nextToken = response.NextToken
   moreData = nextToken != nil
  }
 }
 return output, err
}
```

Utilizar una instrucción UPDATE para actualizar un elemento.

```
// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie
 that
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(ctx context.Context, movie Movie, rating
 float64) error {
 params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
```

```
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
 }
 return err
}
```

Utilizar una instrucción DELETE para eliminar un elemento.

```
// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the
 DynamoDB table.
func (runner PartiQLRunner) DeleteMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

Defina una estructura Película para utilizar en este ejemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
```

```
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obtener información sobre la API, consulte ExecuteStatement en la referencia de la API de AWS SDK para Go.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear un elemento con PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: `INSERT INTO Flowers value {'Name':?}`,
    Parameters: ["Rose"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Obtener un elemento con PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "SELECT * FROM CloudTypes WHERE IsStorm=?",
    Parameters: [false],
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Actualizar un elemento con PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "UPDATE EyeColors SET IsRecessive=? where Color=?",
    Parameters: [true, "blue"],
  });
```

```
  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Eliminar un elemento con PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "DELETE FROM PaintColors where Name=?",
    Parameters: ["Purple"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obtener información sobre la API, consulte ExecuteStatement en la referencia de la API de AWS SDK para JavaScript.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    public function insertItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => "$statement",
            'Parameters' => $parameters,
        ]);
    }

    public function getItemByPartiQL(string $tableName, array $key): Result
    {
        list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);

        return $this->dynamoDbClient->executeStatement([
            'Parameters' => $parameters,
            'Statement' => $statement,
        ]);
    }

    public function updateItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }

    public function deleteItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }
```

- Para obtener información sobre la API, consulte ExecuteStatement en la referencia de la API de AWS SDK para PHP.

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class PartiQLWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statement, params):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statement: The PartiQL statement.
        :param params: The list of PartiQL parameters. These are applied to the
                       statement in the order they are listed.
        :return: The items returned from the statement, if any.
        """
        try:
            output = self.dyn_resource.meta.client.execute_statement(
                Statement=statement, Parameters=params
            )
        except ClientError as err:
```

```
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute PartiQL '%s' because the table does not
 exist.",
                    statement,
                )
            else:
                logger.error(
                    "Couldn't execute PartiQL '%s'. Here's why: %s: %s",
                    statement,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

- Para obtener información sobre la API, consulte ExecuteStatement en Referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Seleccionar un solo elemento con PartiQL.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
```

```
  end

  # Gets a single record from a table using PartiQL.
  # Note: To perform more fine-grained selects,
  # use the Client.query instance method instead.
  #
  # @param title [String] The title of the movie to search.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def select_item_by_title(title)
    request = {
      statement: "SELECT * FROM \"#{@table.name}\" WHERE title=?",
      parameters: [title]
    }
    @dynamodb.client.execute_statement(request)
  end
```

Actualizar un solo elemento con PartiQL.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Updates a single record from a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
  # @param rating [Float] The new rating to assign the title.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def update_rating_by_title(title, year, rating)
    request = {
      statement: "UPDATE \"#{@table.name}\" SET info.rating=? WHERE title=? and
 year=?",
      parameters: [{ "N": rating }, title, year]
    }
    @dynamodb.client.execute_statement(request)
  end
```

Añadir un solo elemento con PartiQL.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Adds a single record to a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
  # @param plot [String] The plot of the movie.
  # @param rating [Float] The new rating to assign the title.
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def insert_item(title, year, plot, rating)
    request = {
      statement: "INSERT INTO \"#{@table.name}\" VALUE {'title': ?, 'year': ?,
 'info': ?}",
      parameters: [title, year, { 'plot': plot, 'rating': rating }]
    }
    @dynamodb.client.execute_statement(request)
  end
```

Eliminar un solo elemento con PartiQL.

```
class DynamoDBPartiQLSingle
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamodb = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamodb.table(table_name)
  end

  # Deletes a single record from a table using PartiQL.
  #
  # @param title [String] The title of the movie to update.
  # @param year [Integer] The year the movie was released.
```

```
  # @return [Aws::DynamoDB::Types::ExecuteStatementOutput]
  def delete_item_by_title(title, year)
    request = {
      statement: "DELETE FROM \"#{@table.name}\" WHERE title=? and year=?",
      parameters: [title, year]
    }
    @dynamodb.client.execute_statement(request)
  end
```

- Para obtener información sobre la API, consulte ExecuteStatement en la referencia de la API de AWS SDK para Ruby.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de **`GetItem`** con un SDK de AWS o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `GetItem`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Conceptos básicos
- Aceleración de lecturas con DAX
- Creación y administración de tablas globales de MRSC
- Trabajo con tablas globales y coherencia final de replicación de varias regiones (MREC)

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</
param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
 newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine($"An error occurred while getting item.
 {ex.Message}");
            throw;
        }
    }
```

- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_get_item
#
# This function gets an item from a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to get.
#       [-q query]  -- Optional JMESPath query expression.
#
#  Returns:
#       The item as text output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_get_item() {
  local table_name keys query response
  local option OPTARG # Required to use getopts command in a function.
```

```
 # ######################################
 # Function usage explanation
 ######################################
 function usage() {
   echo "function dynamodb_get_item"
   echo "Get an item from a DynamoDB table."
   echo " -n table_name  -- The name of the table."
   echo " -k keys  -- Path to json file containing the keys that identify the
item to get."
   echo " [-q query]  -- Optional JMESPath query expression."
   echo ""
 }
 query=""
 while getopts "n:k:q:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) keys="${OPTARG}" ;;
     q) query="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$keys" ]]; then
   errecho "ERROR: You must provide a keys json file path the -k parameter."
   usage
   return 1
 fi
```

```
  if [[ -n "$query" ]]; then
    response=$(aws dynamodb get-item \
      --table-name "$table_name" \
      --key file://"$keys" \
      --output text \
      --query "$query")
  else
    response=$(
      aws dynamodb get-item \
        --table-name "$table_name" \
        --key file://"$keys" \
        --output text
    )
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports get-item operation failed.$response"
    return 1
  fi

  if [[ -n "$query" ]]; then
    echo "$response" | sed "/^\t/s/\t//1" # Remove initial tab that the JMSEPath
 query inserts on some strings.
  else
    echo "$response"
  fi

  return 0
}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
```

```
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información sobre la API, consulte GetItem en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Get an item from an Amazon DynamoDB table.
/*!
  \sa getItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::getItem(const Aws::String &tableName,
                               const Aws::String &partitionKey,
                               const Aws::String &partitionValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::GetItemRequest request;

    // Set up the request.
    request.SetTableName(tableName);
    request.AddKey(partitionKey,
                   Aws::DynamoDB::Model::AttributeValue().SetS(partitionValue));

    // Retrieve the item's fields and values.
    const Aws::DynamoDB::Model::GetItemOutcome &outcome =
 dynamoClient.GetItem(request);
    if (outcome.IsSuccess()) {
        // Reference the retrieved fields/values.
        const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> &item =
 outcome.GetResult().GetItem();
        if (!item.empty()) {
            // Output each retrieved field and its value.
```

```
            for (const auto &i: item)
                std::cout << "Values: " << i.first << ": " << i.second.GetS()
                          << std::endl;
        }
        else {
            std::cout << "No item found with the key " << partitionKey <<
 std::endl;
        }
    }
    else {
        std::cout << "Failed to get item: " << outcome.GetError().GetMessage();
    }

    return outcome.IsSuccess();
}
```

- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Ejemplo 1: Lectura de un elemento de una tabla

En el siguiente ejemplo `get-item`, se recupera un elemento de la tabla `MusicCollection`. La tabla tiene una clave principal hash y de rango (`Artist` y `SongTitle`), por lo que debe especificar ambos atributos. El comando también solicita información sobre la capacidad de lectura consumida por la operación.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --return-consumed-capacity TOTAL
```

Contenido de `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
```

```
}
```

Salida:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Para obtener más información, consulte Lectura de un elemento en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 2: Lectura de un elemento mediante una lectura coherente

En el siguiente ejemplo, se recupera un elemento de la tabla `MusicCollection` con lecturas altamente coherentes.

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json \
    --consistent-read \
    --return-consumed-capacity TOTAL
```

Contenido de `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Salida:

```
{
    "Item": {
        "AlbumTitle": {
            "S": "Songs About Life"
        },
        "SongTitle": {
            "S": "Happy Day"
        },
        "Artist": {
            "S": "Acme Band"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Para obtener más información, consulte Lectura de un elemento en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 3: recuperación de atributos específicos de un elemento

En el siguiente ejemplo, se utiliza una expresión de proyección para recuperar solo tres atributos del elemento deseado.

```
aws dynamodb get-item \
    --table-name ProductCatalog \
    --key '{"Id": {"N": "102"}}' \
    --projection-expression "#T, #C, #P" \
    --expression-attribute-names file://names.json
```

Contenido de `names.json`:

```
{
    "#T": "Title",
    "#C": "ProductCategory",
    "#P": "Price"
}
```

Salida:

```
{
    "Item": {
        "Price": {
            "N": "20"
        },
        "Title": {
            "S": "Book 102 Title"
        },
        "ProductCategory": {
            "S": "Book"
        }
    }
}
```

Para obtener más información, consulte Lectura de un elemento en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte GetItem en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
```

```
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// GetMovie gets movie data from the DynamoDB table by using the primary
 composite key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

Defina una estructura Película para utilizar en este ejemplo.

```
import (
 "archive/zip"
```

```
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Obtiene un elemento de una tabla mediante DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To get an item from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, see the EnhancedGetItem example.
 */
public class GetItem {
    public static void main(String[] args) {
        final String usage = """
```

```
                Usage:
                    <tableName> <key> <keyVal>

                Where:
                    tableName - The Amazon DynamoDB table from which an item is
retrieved (for example, Music3).\s
                    key - The key used in the Amazon DynamoDB table (for example,
Artist).\s
                    keyval - The key value that represents the item to get (for
example, Famous Band).
                """;

       if (args.length != 3) {
           System.out.println(usage);
           System.exit(1);
       }

       String tableName = args[0];
       String key = args[1];
       String keyVal = args[2];
       System.out.format("Retrieving item \"%s\" from \"%s\"\n", keyVal,
tableName);
       Region region = Region.US_EAST_1;
       DynamoDbClient ddb = DynamoDbClient.builder()
               .region(region)
               .build();

       getDynamoDBItem(ddb, tableName, key, keyVal);
       ddb.close();
   }

   public static void getDynamoDBItem(DynamoDbClient ddb, String tableName,
String key, String keyVal) {
       HashMap<String, AttributeValue> keyToGet = new HashMap<>();
       keyToGet.put(key, AttributeValue.builder()
               .s(keyVal)
               .build());

       GetItemRequest request = GetItemRequest.builder()
               .key(keyToGet)
               .tableName(tableName)
               .build();
```

```
        try {
            // If there is no matching item, GetItem does not return any data.
            Map<String, AttributeValue> returnedItem =
 ddb.getItem(request).item();
            if (returnedItem.isEmpty())
                System.out.format("No item found with the key %s!\n", key);
            else {
                Set<String> keys = returnedItem.keySet();
                System.out.println("Amazon DynamoDB table attributes: \n");
                for (String key1 : keys) {
                    System.out.format("%s: %s\n", key1,
 returnedItem.get(key1).toString());
                }
            }

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener información sobre la API, consulte GetCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";
```

```
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Obtener un elemento de una tabla.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Key: {
    KEY_NAME: { N: "001" },
```

```
  },
  ProjectionExpression: "ATTRIBUTE_NAME",
};

// Call DynamoDB to read the item from the table
ddb.getItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

Obtener un elemento de una tabla con el cliente de documentos de DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "EPISODES_TABLE",
  Key: { KEY_NAME: VALUE },
};

docClient.get(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Item);
  }
});
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun getSpecificItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
) {
    val keyToGet = mutableMapOf<String, AttributeValue>()
    keyToGet[keyName] = AttributeValue.S(keyVal)

    val request =
        GetItemRequest {
            key = keyToGet
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val returnedItem = ddb.getItem(request)
        val numbersMap = returnedItem.item
        numbersMap?.forEach { key1 ->
            println(key1.key)
            println(key1.value)
        }
    }
}
```

- Para obtener información de la API, consulte GetItem en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        $movie = $service->getItemByKey($tableName, $key);
        echo "\nThe movie {$movie['Item']['title']['S']} was released in
{$movie['Item']['year']['N']}.\n";

    public function getItemByKey(string $tableName, array $key)
    {
        return $this->dynamoDbClient->getItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
        ]);
    }
```

- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK para PHP.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: devuelve el elemento de DynamoDB con la clave de partición SongTitle y la clave de clasificación Artist.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Salida:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- Para obtener detalles de la API, consulte GetItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: devuelve el elemento de DynamoDB con la clave de partición SongTitle y la clave de clasificación Artist.

```
$key = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
} | ConvertTo-DDBItem

Get-DDBItem -TableName 'Music' -Key $key | ConvertFrom-DDBItem
```

Salida:

```
Name                           Value
----                           -----
Genre                          Country
SongTitle                      Somewhere Down The Road
Price                          1.94
Artist                         No One You Know
CriticRating                   9
AlbumTitle                     Somewhat Famous
```

- Para obtener detalles de la API, consulte GetItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def get_movie(self, title, year):
        """
        Gets movie data from the table for a specific movie.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :return: The data about the requested movie.
        """
        try:
            response = self.table.get_item(Key={"year": year, "title": title})
        except ClientError as err:
            logger.error(
                "Couldn't get movie %s from table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Item"]
```

- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table
```

```
def initialize(table_name)
  client = Aws::DynamoDB::Client.new(region: 'us-east-1')
  @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
  @table = @dynamo_resource.table(table_name)
end

# Gets movie data from the table for a specific movie.
#
# @param title [String] The title of the movie.
# @param year [Integer] The release year of the movie.
# @return [Hash] The data about the requested movie.
def get_item(title, year)
  @table.get_item(key: { 'year' => year, 'title' => title })
rescue Aws::DynamoDB::Errors::ServiceError => e
  puts("Couldn't get movie #{title} (#{year}) from table #{@table.name}:\n")
  puts("\t#{e.code}: #{e.message}")
  raise
end
```

- Para obtener información sobre la API, consulte GetItem en la referencia de la API de AWS SDK para Ruby.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
TRY.
    oo_item = lo_dyn->getitem(
      iv_tablename               = iv_table_name
      it_key                     = it_key ).
    DATA(lt_attr) = oo_item->get_item( ).
    DATA(lo_title) = lt_attr[ key = 'title' ]-value.
    DATA(lo_year) = lt_attr[ key = 'year' ]-value.
    DATA(lo_rating) = lt_attr[ key = 'rating' ]-value.
```

```
        MESSAGE 'Movie name is: ' && lo_title->get_s( )
          && 'Movie year is: ' && lo_year->get_n( )
          && 'Moving rating is: ' && lo_rating->get_n( ) TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Para obtener información sobre la API, consulte GetItem en la Referencia de la API de AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB


    /// Return a `Movie` record describing the specified movie from the Amazon
    /// DynamoDB table.
    ///
    /// - Parameters:
    ///   - title: The movie's title (`String`).
    ///   - year: The movie's release year (`Int`).
    ///
    /// - Throws: `MoviesError.ItemNotFound` if the movie isn't in the table.
    ///
    /// - Returns: A `Movie` record with the movie's details.
    func get(title: String, year: Int) async throws -> Movie {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = GetItemInput(
```

```
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                tableName: self.tableName
            )
            let output = try await client.getItem(input: input)
            guard let item = output.item else {
                throw MoviesError.ItemNotFound
            }

            let movie = try Movie(withItem: item)
            return movie
        } catch {
            print("ERROR: get:", dump(error))
            throw error
        }
    }
```

- Para obtener detalles sobre la API, consulte GetItem en la Referencia de la API del SDK de AWS para Swift.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de **`ListTables`** con un SDK de AWS o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `ListTables`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- Configuración del control de acceso basado en atributos

.NET

SDK para .NET

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
private static async Task ListMyTables()
{
    Console.WriteLine("\n*** Listing tables ***");

    string lastTableNameEvaluated = null;
    do
    {
        var response = await Client.ListTablesAsync(new ListTablesRequest
        {
            Limit = 2,
            ExclusiveStartTableName = lastTableNameEvaluated
        });

        foreach (var name in response.TableNames)
        {
            Console.WriteLine(name);
        }

        lastTableNameEvaluated = response.LastEvaluatedTableName;
    } while (lastTableNameEvaluated != null);
}
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_list_tables
#
# This function lists all the tables in a DynamoDB.
#
# Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_list_tables() {
  response=$(aws dynamodb list-tables \
    --output text \
    --query "TableNames")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports batch-write-item operation failed.$response"
    return 1
  fi

  echo "$response" | tr -s "[:space:]" "\n"

  return 0
}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function errecho
```

```
#
# This function outputs everything sent to it to STDERR (standard error output).
#####################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#####################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#####################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información sobre la API, consulte ListTables en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! List the Amazon DynamoDB tables for the current AWS account.
/*!
  \sa listTables()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::listTables(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::ListTablesRequest listTablesRequest;
    listTablesRequest.SetLimit(50);
    do {
        const Aws::DynamoDB::Model::ListTablesOutcome &outcome =
 dynamoClient.ListTables(
                listTablesRequest);
        if (!outcome.IsSuccess()) {
            std::cout << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
            return false;
        }

        for (const auto &tableName: outcome.GetResult().GetTableNames())
            std::cout << tableName << std::endl;
        listTablesRequest.SetExclusiveStartTableName(
```

```
                outcome.GetResult().GetLastEvaluatedTableName());

    } while (!listTablesRequest.GetExclusiveStartTableName().empty());

    return true;
}
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Ejemplo 1: Creación de una lista de tablas

En el siguiente ejemplo `list-tables`, se enumeran las tablas asociadas a la cuenta y región de AWS.

```
aws dynamodb list-tables
```

Salida:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

Para obtener más información, consulte Enumeración de nombres de tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 2: Limitación del tamaño de la página

En el siguiente ejemplo, se devuelve una lista de todas las tablas existentes, pero recupera solo un elemento en cada llamada y, si es necesario, realiza varias llamadas para obtener la lista completa. Limitar el tamaño de página resulta útil cuando se ejecutan comandos de la

lista en un gran número de recursos, lo que puede provocar un error de “tiempo de espera” cuando se utiliza el tamaño de página predeterminado de 1000.

```
aws dynamodb list-tables \
    --page-size 1
```

Salida:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog",
        "Reply",
        "Thread"
    ]
}
```

Para obtener más información, consulte Enumeración de nombres de tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 3: limitación del número de elementos devueltos

En el siguiente ejemplo, se limita el número de elementos devueltos a 2. La respuesta incluye un valor `NextToken` con el que recuperar la siguiente página de resultados.

```
aws dynamodb list-tables \
    --max-items 2
```

Salida:

```
{
    "TableNames": [
        "Forum",
        "ProductCatalog"
    ],
    "NextToken":
 "abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9"
}
```

Para obtener más información, consulte Enumeración de nombres de tablas en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 4: recuperación de la siguiente página de resultados

El comando siguiente utiliza el valor `NextToken` de una llamada anterior al comando `list-tables` para recuperar otra página de resultados. Puesto que la respuesta en este caso no incluye ningún valor `NextToken`, sabemos que hemos llegado al final de los resultados.

```
aws dynamodb list-tables \
    --starting-
token abCDeFGhiJKlmnOPqrSTuvwxYZ1aBCdEFghijK7LM51nOpqRSTuv3WxY3ZabC5dEFGhI2Jk3LmnoPQ6RST9
```

Salida:

```
{
    "TableNames": [
        "Reply",
        "Thread"
    ]
}
```

Para obtener más información, consulte Enumeración de nombres de tablas en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte ListTables en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
 "errors"
 "log"
```

```
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// ListTables lists the DynamoDB table names for the current account.
func (basics TableBasics) ListTables(ctx context.Context) ([]string, error) {
 var tableNames []string
 var output *dynamodb.ListTablesOutput
 var err error
 tablePaginator := dynamodb.NewListTablesPaginator(basics.DynamoDbClient,
 &dynamodb.ListTablesInput{})
 for tablePaginator.HasMorePages() {
  output, err = tablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't list tables. Here's why: %v\n", err)
   break
  } else {
   tableNames = append(tableNames, output.TableNames...)
  }
 }
 return tableNames, err
}
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListTables {
    public static void main(String[] args) {
        System.out.println("Listing your Amazon DynamoDB tables:\n");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        listAllTables(ddb);
        ddb.close();
    }

    public static void listAllTables(DynamoDbClient ddb) {
        boolean moreTables = true;
        String lastName = null;

        while (moreTables) {
```

```
            try {
                ListTablesResponse response = null;
                if (lastName == null) {
                    ListTablesRequest request =
 ListTablesRequest.builder().build();
                    response = ddb.listTables(request);
                } else {
                    ListTablesRequest request = ListTablesRequest.builder()
                            .exclusiveStartTableName(lastName).build();
                    response = ddb.listTables(request);
                }

                List<String> tableNames = response.tableNames();
                if (tableNames.size() > 0) {
                    for (String curName : tableNames) {
                        System.out.format("* %s\n", curName);
                    }
                } else {
                    System.out.println("No tables found!");
                    System.exit(0);
                }

                lastName = response.lastEvaluatedTableName();
                if (lastName == null) {
                    moreTables = false;
                }

            } catch (DynamoDbException e) {
                System.err.println(e.getMessage());
                System.exit(1);
            }
        }
        System.out.println("\nDone!");
    }
}
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

// Call DynamoDB to retrieve the list of tables
ddb.listTables({ Limit: 10 }, function (err, data) {
  if (err) {
    console.log("Error", err.code);
  } else {
    console.log("Table names are ", data.TableNames);
  }
});
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun listAllTables() {
    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.listTables(ListTablesRequest {})
        response.tableNames?.forEach { tableName ->
            println("Table name is $tableName")
        }
    }
}
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    public function listTables($exclusiveStartTableName = "", $limit = 100)
    {
        $this->dynamoDbClient->listTables([
            'ExclusiveStartTableName' => $exclusiveStartTableName,
            'Limit' => $limit,
        ]);
    }
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para PHP.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: devuelve los detalles de todas las tablas y se itera automáticamente hasta que el servicio indique que no existen más tablas.

```
Get-DDBTableList
```

- Para obtener detalles de la API, consulte ListTables en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: devuelve los detalles de todas las tablas y se itera automáticamente hasta que el servicio indique que no existen más tablas.

```
Get-DDBTableList
```

- Para obtener detalles de la API, consulte ListTables en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
```

```
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def list_tables(self):
        """
        Lists the Amazon DynamoDB tables for the current account.

        :return: The list of tables.
        """
        try:
            tables = []
            for table in self.dyn_resource.tables.all():
                print(table.name)
                tables.append(table)
        except ClientError as err:
            logger.error(
                "Couldn't list tables. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return tables
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Determinar si existe una tabla.

```
# Encapsulates an Amazon DynamoDB table of movie data.
class Scaffold
  attr_reader :dynamo_resource, :table_name, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table_name = table_name
    @table = nil
    @logger = Logger.new($stdout)
    @logger.level = Logger::DEBUG
  end

  # Determines whether a table exists. As a side effect, stores the table in
  # a member variable.
  #
  # @param table_name [String] The name of the table to check.
  # @return [Boolean] True when the table exists; otherwise, False.
  def exists?(table_name)
    @dynamo_resource.client.describe_table(table_name: table_name)
    @logger.debug("Table #{table_name} exists")
  rescue Aws::DynamoDB::Errors::ResourceNotFoundException
    @logger.debug("Table #{table_name} doesn't exist")
    false
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't check for existence of #{table_name}:\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
pub async fn list_tables(client: &Client) -> Result<Vec<String>, Error> {
    let paginator = client.list_tables().into_paginator().items().send();
    let table_names = paginator.collect::<Result<Vec<_>, _>>().await?;

    println!("Tables:");

    for name in &table_names {
        println!("  {}", name);
    }

    println!("Found {} tables", table_names.len());
    Ok(table_names)
}
```

Determinar si existe una tabla.

```
pub async fn table_exists(client: &Client, table: &str) -> Result<bool, Error> {
    debug!("Checking for table: {table}");
    let table_list = client.list_tables().send().await;

    match table_list {
        Ok(list) => Ok(list.table_names().contains(&table.into())),
        Err(e) => Err(e.into()),
    }
}
```

- Para obtener información sobre la API, consulte ListTables en la referencia de la API de AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    TRY.
        oo_result = lo_dyn->listtables( ).
        " You can loop over the oo_result to get table properties like this.
        LOOP AT oo_result->get_tablenames( ) INTO DATA(lo_table_name).
          DATA(lv_tablename) = lo_table_name->get_value( ).
        ENDLOOP.
        DATA(lv_tablecount) = lines( oo_result->get_tablenames( ) ).
        MESSAGE 'Found ' && lv_tablecount && ' tables' TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_exception).
        DATA(lv_error) = |"{ lo_exception->av_err_code }" - { lo_exception-
>av_err_msg }|.
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- Para obtener información sobre la API, consulte ListTables en la referencia de API del SDK AWSpara SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB
```

```
/// Get a list of the DynamoDB tables available in the specified Region.
///
/// - Returns: An array of strings listing all of the tables available
///   in the Region specified when the session was created.
public func getTableList() async throws -> [String] {
    let input = ListTablesInput(
    )
    return try await session.listTables(input: input)
}
```

- Para obtener información acerca de la API, consulte ListTables en la Referencia de la API del AWS SDK para Swift.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de **`PutItem`** con un SDK de AWS o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `PutItem`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Conceptos básicos
- Aceleración de lecturas con DAX
- Escenarios avanzados de índice secundario global
- Creación de un elemento con un TTL
- Creación y administración de tablas globales que muestren MREC
- Creación y administración de tablas globales de MRSC
- Uso de operaciones condicionales
- Trabajo con tablas globales y coherencia final de replicación de varias regiones (MREC)
- Trabajo con Streams y tiempo de vida

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
```

```
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_put_item
#
# This function puts an item into a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -i item  -- Path to json file containing the item values.
#
#  Returns:
```

```
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_put_item() {
  local table_name item response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_put_item"
    echo "Put an item into a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -i item  -- Path to json file containing the item values."
    echo ""
  }

  while getopts "n:i:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      i) item="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$item" ]]; then
    errecho "ERROR: You must provide an item with the -i parameter."
    usage
```

```
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    item:   $item"
  iecho ""
  iecho ""

  response=$(aws dynamodb put-item \
    --table-name "$table_name" \
    --item file://"$item")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports put-item operation failed.$response"
    return 1
  fi

  return 0

}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
```

```
# This function outputs everything sent to it to STDERR (standard error output).
#########################################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

#########################################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
#########################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información sobre la API, consulte PutItem en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Put an item in an Amazon DynamoDB table.
/*!
  \sa putItem()
  \param tableName: The table name.
  \param artistKey: The artist key. This is the partition key for the table.
  \param artistValue: The artist value.
  \param albumTitleKey: The album title key.
  \param albumTitleValue: The album title value.
  \param awardsKey: The awards key.
  \param awardsValue: The awards value.
  \param songTitleKey: The song title key.
  \param songTitleValue: The song title value.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::putItem(const Aws::String &tableName,
                               const Aws::String &artistKey,
                               const Aws::String &artistValue,
                               const Aws::String &albumTitleKey,
                               const Aws::String &albumTitleValue,
                               const Aws::String &awardsKey,
                               const Aws::String &awardsValue,
                               const Aws::String &songTitleKey,
                               const Aws::String &songTitleValue,
                               const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
```

```
    Aws::DynamoDB::Model::PutItemRequest putItemRequest;
    putItemRequest.SetTableName(tableName);

    putItemRequest.AddItem(artistKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            artistValue)); // This is the hash key.
    putItemRequest.AddItem(albumTitleKey,
 Aws::DynamoDB::Model::AttributeValue().SetS(
            albumTitleValue));
    putItemRequest.AddItem(awardsKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(awardsValue));
    putItemRequest.AddItem(songTitleKey,

 Aws::DynamoDB::Model::AttributeValue().SetS(songTitleValue));

    const Aws::DynamoDB::Model::PutItemOutcome outcome = dynamoClient.PutItem(
            putItemRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully added Item!" << std::endl;
    }
    else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Código que espera a que se active la tabla.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {
```

```
    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Ejemplo 1: Adición de un elemento a una tabla

En el siguiente ejemplo `put-item`, se añade un elemento nuevo a la tabla MusicCollection.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Contenido de `item.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Greatest Hits"}
}
```

Salida:

```
{
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "No One You Know"
            }
        },
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Para obtener más información, consulte Escritura de un elemento en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 2: Sobrescritura condicional de un elemento de una tabla

En el siguiente ejemplo de `put-item` se sobrescribe un elemento existente de la tabla `MusicCollection` solo si ese elemento existente tiene un atributo `AlbumTitle` con un valor de `Greatest Hits`. El comando devuelve el valor anterior del elemento.

```
aws dynamodb put-item \
    --table-name MusicCollection \
    --item file://item.json \
    --condition-expression "#A = :A" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-values ALL_OLD
```

Contenido de `item.json`:

```
{
    "Artist": {"S": "No One You Know"},
    "SongTitle": {"S": "Call Me Today"},
    "AlbumTitle": {"S": "Somewhat Famous"}
}
```

Contenido de `names.json`:

```
{
    "#A": "AlbumTitle"
}
```

Contenido de `values.json`:

```
{
    ":A": {"S": "Greatest Hits"}
}
```

Salida:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Greatest Hits"
        },
        "Artist": {
            "S": "No One You Know"
        },
        "SongTitle": {
            "S": "Call Me Today"
```

```
        }
    }
}
```

Si la clave ya existe, debería ver el siguiente resultado:

```
A client error (ConditionalCheckFailedException) occurred when calling the
 PutItem operation: The conditional request failed.
```

Para obtener más información, consulte Escritura de un elemento en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte PutItem en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
```

```
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}
```

Defina una estructura Película para utilizar en este ejemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)
```

```
// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Coloca un elemento en una tabla mediante DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To place items into an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedPutItem example.
 */
public class PutItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <albumtitle> <albumtitleval>
 <awards> <awardsval> <Songtitle> <songtitleval>

                Where:
```

```
                    tableName - The Amazon DynamoDB table in which an item is
placed (for example, Music3).
                    key - The key used in the Amazon DynamoDB table (for example,
Artist).
                    keyval - The key value that represents the item to get (for
example, Famous Band).
                    albumTitle - The Album title (for example, AlbumTitle).
                    AlbumTitleValue - The name of the album (for example, Songs
About Life ).
                    Awards - The awards column (for example, Awards).
                    AwardVal - The value of the awards (for example, 10).
                    SongTitle - The song title (for example, SongTitle).
                    SongTitleVal - The value of the song title (for example,
Happy Day).
                **Warning** This program will  place an item that you specify
into a table!
                """;

        if (args.length != 9) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String albumTitle = args[3];
        String albumTitleValue = args[4];
        String awards = args[5];
        String awardVal = args[6];
        String songTitle = args[7];
        String songTitleVal = args[8];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        putItemInTable(ddb, tableName, key, keyVal, albumTitle, albumTitleValue,
awards, awardVal, songTitle,
                songTitleVal);
        System.out.println("Done!");
        ddb.close();
    }
```

```
    public static void putItemInTable(DynamoDbClient ddb,
            String tableName,
            String key,
            String keyVal,
            String albumTitle,
            String albumTitleValue,
            String awards,
            String awardVal,
            String songTitle,
            String songTitleVal) {

        HashMap<String, AttributeValue> itemValues = new HashMap<>();
        itemValues.put(key, AttributeValue.builder().s(keyVal).build());
        itemValues.put(songTitle,
 AttributeValue.builder().s(songTitleVal).build());
        itemValues.put(albumTitle,
 AttributeValue.builder().s(albumTitleValue).build());
        itemValues.put(awards, AttributeValue.builder().s(awardVal).build());

        PutItemRequest request = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();

        try {
            PutItemResponse response = ddb.putItem(request);
            System.out.println(tableName + " was successfully updated. The
 request id is "
                    + response.responseMetadata().requestId());

        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            System.err.println("Be sure that it exists and that you've typed its
 name correctly!");
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener información sobre la API, consulte PutCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Colocar un elemento en una tabla.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the DynamoDB service object
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

var params = {
  TableName: "CUSTOMER_LIST",
  Item: {
    CUSTOMER_ID: { N: "001" },
    CUSTOMER_NAME: { S: "Richard Roe" },
  },
};

// Call DynamoDB to add the item to the table
ddb.putItem(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

Colocar un elemento en una tabla con el cliente de documentos de DynamoDB.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });
```

```
// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  TableName: "TABLE",
  Item: {
    HASHKEY: VALUE,
    ATTRIBUTE_1: "STRING_VALUE",
    ATTRIBUTE_2: VALUE_2,
  },
};

docClient.put(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun putItemInTable(
    tableNameVal: String,
    key: String,
    keyVal: String,
```

```
    albumTitle: String,
    albumTitleValue: String,
    awards: String,
    awardVal: String,
    songTitle: String,
    songTitleVal: String,
) {
    val itemValues = mutableMapOf<String, AttributeValue>()

    // Add all content to the table.
    itemValues[key] = AttributeValue.S(keyVal)
    itemValues[songTitle] = AttributeValue.S(songTitleVal)
    itemValues[albumTitle] = AttributeValue.S(albumTitleValue)
    itemValues[awards] = AttributeValue.S(awardVal)

    val request =
        PutItemRequest {
            tableName = tableNameVal
            item = itemValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.putItem(request)
        println(" A new item was placed into $tableNameVal.")
    }
}
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }

        $service->putItem([
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
            'TableName' => $tableName,
        ]);

    public function putItem(array $array)
    {
        $this->dynamoDbClient->putItem($array);
    }
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para PHP.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: crea un nuevo elemento o sustituye un elemento existente por uno nuevo.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
```

```
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- Para obtener detalles de la API, consulte PutItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: crea un nuevo elemento o sustituye un elemento existente por uno nuevo.

```
$item = @{
  SongTitle = 'Somewhere Down The Road'
  Artist = 'No One You Know'
        AlbumTitle = 'Somewhat Famous'
        Price = 1.94
        Genre = 'Country'
        CriticRating = 9.0
} | ConvertTo-DDBItem
Set-DDBItem -TableName 'Music' -Item $item
```

- Para obtener detalles de la API, consulte PutItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
```

```
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def add_movie(self, title, year, plot, rating):
        """
        Adds a movie to the table.

        :param title: The title of the movie.
        :param year: The release year of the movie.
        :param plot: The plot summary of the movie.
        :param rating: The quality rating of the movie.
        """
        try:
            self.table.put_item(
                Item={
                    "year": year,
                    "title": title,
                    "info": {"plot": plot, "rating": Decimal(str(rating))},
                }
            )
        except ClientError as err:
```

```
            logger.error(
                "Couldn't add movie %s to table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Adds a movie to the table.
  #
  # @param movie [Hash] The title, year, plot, and rating of the movie.
  def add_item(movie)
    @table.put_item(
      item: {
        'year' => movie[:year],
        'title' => movie[:title],
```

```
        'info' => { 'plot' => movie[:plot], 'rating' => movie[:rating] }
      }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't add movie #{title} to table #{@table.name}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  end
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
pub async fn add_item(client: &Client, item: Item, table: &String) ->
 Result<ItemOut, Error> {
    let user_av = AttributeValue::S(item.username);
    let type_av = AttributeValue::S(item.p_type);
    let age_av = AttributeValue::S(item.age);
    let first_av = AttributeValue::S(item.first);
    let last_av = AttributeValue::S(item.last);

    let request = client
        .put_item()
        .table_name(table)
        .item("username", user_av)
        .item("account_type", type_av)
        .item("age", age_av)
        .item("first_name", first_av)
        .item("last_name", last_av);

    println!("Executing request [{request:?}] to add item...");
```

```
    let resp = request.send().await?;

    let attributes = resp.attributes().unwrap();

    let username = attributes.get("username").cloned();
    let first_name = attributes.get("first_name").cloned();
    let last_name = attributes.get("last_name").cloned();
    let age = attributes.get("age").cloned();
    let p_type = attributes.get("p_type").cloned();

    println!(
        "Added user {:?}, {:?} {:?}, age {:?} as {:?} user",
        username, first_name, last_name, age, p_type
    );

    Ok(ItemOut {
        p_type,
        age,
        username,
        first_name,
        last_name,
    })
}
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    TRY.
        DATA(lo_resp) = lo_dyn->putitem(
```

```
          iv_tablename = iv_table_name
          it_item      = it_item ).
        MESSAGE '1 row inserted into DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB


    /// Add a movie specified as a `Movie` structure to the Amazon DynamoDB
    /// table.
    ///
    /// - Parameter movie: The `Movie` to add to the table.
    ///
    func add(movie: Movie) async throws {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Get a DynamoDB item containing the movie data.
```

```
            let item = try await movie.getAsItem()

            // Send the `PutItem` request to Amazon DynamoDB.

            let input = PutItemInput(
                item: item,
                tableName: self.tableName
            )
            _ = try await client.putItem(input: input)
        } catch {
            print("ERROR: add movie:", dump(error))
            throw error
        }
    }


    ///
    /// Return an array mapping attribute names to Amazon DynamoDB attribute
    /// values, representing the contents of the `Movie` record as a DynamoDB
    /// item.
    ///
    /// - Returns: The movie item as an array of type
    ///   `[Swift.String:DynamoDBClientTypes.AttributeValue]`.
    ///
    func getAsItem() async throws ->
[Swift.String:DynamoDBClientTypes.AttributeValue]  {
        // Build the item record, starting with the year and title, which are
        // always present.

        var item: [Swift.String:DynamoDBClientTypes.AttributeValue] = [
            "year": .n(String(self.year)),
            "title": .s(self.title)
        ]

        // Add the `info` field with the rating and/or plot if they're
        // available.

        var details: [Swift.String:DynamoDBClientTypes.AttributeValue] = [:]
        if (self.info.rating != nil || self.info.plot != nil) {
            if self.info.rating != nil {
                details["rating"] = .n(String(self.info.rating!))
            }
            if self.info.plot != nil {
                details["plot"] = .s(self.info.plot!)
```

```
            }
        }
        item["info"] = .m(details)

        return item
    }
```

- Para obtener información acerca de la API, consulte PutItem en la Referencia de la API del SDK de AWS para Swift.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de **`Query`** con un SDK de AWS o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `Query`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Conceptos básicos
- Aceleración de lecturas con DAX
- Escenarios avanzados de índice secundario global
- Comparación de varios valores con un solo atributo
- Administración de índices secundarios globales
- Realización de operaciones de consulta avanzadas
- Consulta de una tabla mediante una condición begins_with
- Consulta de una tabla mediante un intervalo de fechas
- Consulta de una tabla mediante un índice secundario global
- Consulta de una tabla con una expresión de filtro compleja
- Consulta de una tabla con una expresión de filtro dinámica
- Consulta de una tabla con una expresión de filtro y limite
- Consulta de una tabla con atributos anidados
- Consulta de una tabla con paginación

- Consulta de una tabla con lecturas altamente coherentes
- Consulta de elementos de TTL
- Consulta de tablas mediante patrones de fecha y hora
- Uso de nombres de atributo de expresión
- Trabajo con índices secundarios locales

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
            .AddRangeKey("title", DynamoDBEntryType.String)
            .Build();

        var filter = new QueryFilter("year", QueryOperator.Equal, year);

        Console.WriteLine("\nFind movies released in: {year}:");

        var config = new QueryOperationConfig()
        {
```

```
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };

            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_query
#
# This function queries a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k key_condition_expression -- The key condition expression.
#       -a attribute_names -- Path to JSON file containing the attribute names.
#       -v attribute_values -- Path to JSON file containing the attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_query() {
```

```
 local table_name key_condition_expression attribute_names attribute_values
projection_expression response
 local option OPTARG # Required to use getopts command in a function.

 # ######################################
 # Function usage explanation
 ######################################
 function usage() {
   echo "function dynamodb_query"
   echo "Query a DynamoDB table."
   echo " -n table_name  -- The name of the table."
   echo " -k key_condition_expression -- The key condition expression."
   echo " -a attribute_names -- Path to JSON file containing the attribute
names."
   echo " -v attribute_values -- Path to JSON file containing the attribute
values."
   echo " [-p projection_expression]  -- Optional projection expression."
   echo ""
 }

 while getopts "n:k:a:v:p:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) key_condition_expression="${OPTARG}" ;;
     a) attribute_names="${OPTARG}" ;;
     v) attribute_values="${OPTARG}" ;;
     p) projection_expression="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
```

```
 fi

 if [[ -z "$key_condition_expression" ]]; then
   errecho "ERROR: You must provide a key condition expression with the -k
parameter."
   usage
   return 1
 fi

 if [[ -z "$attribute_names" ]]; then
   errecho "ERROR: You must provide a attribute names with the -a parameter."
   usage
   return 1
 fi

 if [[ -z "$attribute_values" ]]; then
   errecho "ERROR: You must provide a attribute values with the -v parameter."
   usage
   return 1
 fi

 if [[ -z "$projection_expression" ]]; then
   response=$(aws dynamodb query \
     --table-name "$table_name" \
     --key-condition-expression "$key_condition_expression" \
     --expression-attribute-names file://"$attribute_names" \
     --expression-attribute-values file://"$attribute_values")
 else
   response=$(aws dynamodb query \
     --table-name "$table_name" \
     --key-condition-expression "$key_condition_expression" \
     --expression-attribute-names file://"$attribute_names" \
     --expression-attribute-values file://"$attribute_values" \
     --projection-expression "$projection_expression")
 fi

 local error_code=${?}

 if [[ $error_code -ne 0 ]]; then
   aws_cli_error_log $error_code
   errecho "ERROR: AWS reports query operation failed.$response"
   return 1
 fi
```

```
  echo "$response"

  return 0
}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
```

```
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información sobre la API, consulte Query en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Perform a query on an Amazon DynamoDB Table and retrieve items.
/*!
  \sa queryItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param projectionExpression: The projections expression, which is ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 * The partition key attribute is searched with the specified value. By default,
 all fields and values
 * contained in the item are returned. If an optional projection expression is
```

```
 * specified on the command line, only the specified fields and values are
 * returned.
 */

bool AwsDoc::DynamoDB::queryItems(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &projectionExpression,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::QueryRequest request;

    request.SetTableName(tableName);

    if (!projectionExpression.empty()) {
        request.SetProjectionExpression(projectionExpression);
    }

    // Set query key condition expression.
    request.SetKeyConditionExpression(partitionKey + "= :valueToMatch");

    // Set Expression AttributeValues.
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue> attributeValues;
    attributeValues.emplace(":valueToMatch", partitionValue);

    request.SetExpressionAttributeValues(attributeValues);

    bool result = true;

    // "exclusiveStartKey" is used for pagination.
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 exclusiveStartKey;
    do {
        if (!exclusiveStartKey.empty()) {
            request.SetExclusiveStartKey(exclusiveStartKey);
            exclusiveStartKey.clear();
        }
        // Perform Query operation.
        const Aws::DynamoDB::Model::QueryOutcome &outcome =
 dynamoClient.Query(request);
        if (outcome.IsSuccess()) {
            // Reference the retrieved items.
```

```
            const Aws::Vector<Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
            if (!items.empty()) {
                std::cout << "Number of items retrieved from Query: " <<
 items.size()
                          << std::endl;
                // Iterate each item and print.
                for (const auto &item: items) {
                    std::cout
                            <<
 "*************************************************"
                            << std::endl;
                    // Output each retrieved field and its value.
                    for (const auto &i: item)
                        std::cout << i.first << ": " << i.second.GetS() <<
 std::endl;
                }
            }
            else {
                std::cout << "No item found in table: " << tableName <<
 std::endl;
            }

            exclusiveStartKey = outcome.GetResult().GetLastEvaluatedKey();
        }
        else {
            std::cerr << "Failed to Query items: " <<
 outcome.GetError().GetMessage();
            result = false;
            break;
        }
    } while (!exclusiveStartKey.empty());

    return result;
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Ejemplo 1: Consulta de una tabla

En el siguiente ejemplo de `query` se consultan elementos de la tabla `MusicCollection`. La tabla tiene una clave principal hash y de rango (`Artist` y `SongTitle`), pero esta consulta solo especifica el valor de la clave hash. Devuelve los títulos de las canciones del artista llamado “No One You Know”.

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --return-consumed-capacity TOTAL
```

Contenido de `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Salida:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
```

```
    }
}
```

Para obtener más información, consulte Uso de consultas en DynamoDB en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 2: Consulta de una tabla con lecturas altamente coherentes y recorrer el índice en orden descendente

En el siguiente ejemplo se realiza la misma consulta que en el primer ejemplo, pero se devuelven los resultados en orden inverso y se utilizan lecturas altamente coherentes.

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --consistent-read \
    --no-scan-index-forward \
    --return-consumed-capacity TOTAL
```

Contenido de `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Salida:

```
{
    "Items": [
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
```

```
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 1.0
    }
}
```

Para obtener más información, consulte Uso de consultas en DynamoDB en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 3: filtrado de resultados específicos

En el siguiente ejemplo se consulta `MusicCollection` pero se excluyen los resultados con valores específicos en el atributo `AlbumTitle`. Tenga en cuenta que esto no afecta a `ScannedCount` o `ConsumedCapacity`, ya que el filtro se aplica después de leer los elementos.

```
aws dynamodb query \
    --table-name MusicCollection \
    --key-condition-expression "#n1 = :v1" \
    --filter-expression "NOT (#n2 IN (:v2, :v3))" \
    --expression-attribute-names file://names.json \
    --expression-attribute-values file://values.json \
    --return-consumed-capacity TOTAL
```

Contenido de `values.json`:

```
{
    ":v1": {"S": "No One You Know"},
    ":v2": {"S": "Blue Sky Blues"},
    ":v3": {"S": "Greatest Hits"}
}
```

Contenido de `names.json`:

```
{
    "#n1": "Artist",
    "#n2": "AlbumTitle"
}
```

Salida:

```
{
    "Items": [
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 1,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5
    }
}
```

Para obtener más información, consulte Uso de consultas en DynamoDB en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 4: Recuperación de un solo recuento de elementos

En el siguiente ejemplo, se recupera un recuento de los elementos que coinciden con la consulta, pero no recupera ninguno de los elementos en sí.

```
aws dynamodb query \
    --table-name MusicCollection \
    --select COUNT \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

Contenido de `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
```

```
}
```

Salida:

```
{
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": null
}
```

Para obtener más información, consulte Uso de consultas en DynamoDB en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 5: consulta de un índice

El siguiente ejemplo consulta el índice secundario global `AlbumTitleIndex`. La consulta devuelve todos los atributos de la tabla base que se han proyectado en el índice secundario local. Tenga en cuenta que, al consultar un índice secundario local o un índice secundario global, también debe proporcionar el nombre de la tabla base mediante el parámetro `table-name`.

```
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumTitleIndex \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json \
    --select ALL_PROJECTED_ATTRIBUTES \
    --return-consumed-capacity INDEXES
```

Contenido de `expression-attributes.json`:

```
{
    ":v1": {"S": "No One You Know"}
}
```

Salida:

```
{
    "Items": [
```

```
        {
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Scared of My Shadow"
            }
        },
        {
            "AlbumTitle": {
                "S": "Somewhat Famous"
            },
            "Artist": {
                "S": "No One You Know"
            },
            "SongTitle": {
                "S": "Call Me Today"
            }
        }
    ],
    "Count": 2,
    "ScannedCount": 2,
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 0.5,
        "Table": {
            "CapacityUnits": 0.0
        },
        "LocalSecondaryIndexes": {
            "AlbumTitleIndex": {
                "CapacityUnits": 0.5
            }
        }
    }
}
```

Para obtener más información, consulte Uso de consultas en DynamoDB en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte Query en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Query gets all movies in the DynamoDB table that were released in the
 specified year.
// The function uses the `expression` package to build the key condition
 expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
```

```
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

Defina una estructura Película para utilizar en este ejemplo.

```
import (
 "archive/zip"
```

```
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Consultar una tabla con DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To query items from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client. See the EnhancedQueryRecords example.
 */
public class Query {
    public static void main(String[] args) {
        final String usage = """
```

```
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyVal>

                Where:
                    tableName - The Amazon DynamoDB table to put the item in (for
 example, Music3).
                    partitionKeyName - The partition key name of the Amazon
 DynamoDB table (for example, Artist).
                    partitionKeyVal - The value of the partition key that should
 match (for example, Famous Band).
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String partitionKeyName = args[1];
        String partitionKeyVal = args[2];

        // For more information about an alias, see:
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Expressions.ExpressionAttributeNames.html
        String partitionAlias = "#a";

        System.out.format("Querying %s", tableName);
        System.out.println("");
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        int count = queryTable(ddb, tableName, partitionKeyName, partitionKeyVal,
 partitionAlias);
        System.out.println("There were " + count + "  record(s) returned");
        ddb.close();
    }

    public static int queryTable(DynamoDbClient ddb, String tableName, String
 partitionKeyName, String partitionKeyVal,
            String partitionAlias) {
        // Set up an alias for the partition key name in case it's a reserved
 word.
```

```
        HashMap<String, String> attrNameAlias = new HashMap<String, String>();
        attrNameAlias.put(partitionAlias, partitionKeyName);

        // Set up mapping of the partition name with the value.
        HashMap<String, AttributeValue> attrValues = new HashMap<>();
        attrValues.put(":" + partitionKeyName, AttributeValue.builder()
                .s(partitionKeyVal)
                .build());

        QueryRequest queryReq = QueryRequest.builder()
                .tableName(tableName)
                .keyConditionExpression(partitionAlias + " = :" +
 partitionKeyName)
                .expressionAttributeNames(attrNameAlias)
                .expressionAttributeValues(attrValues)
                .build();

        try {
            QueryResponse response = ddb.query(queryReq);
            return response.count();

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return -1;
    }
}
```

Consultar una tabla con `DynamoDbClient` y un índice secundario.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import java.util.HashMap;
import java.util.Map;

/**
 * Before running this Java V2 code example, set up your development
```

```
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * Create the Movies table by running the Scenario example and loading the Movie
 * data from the JSON file. Next create a secondary
 * index for the Movies table that uses only the year column. Name the index
 * **year-index**. For more information, see:
 *
 * https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html
 */
public class QueryItemsUsingIndex {
    public static void main(String[] args) {
        String tableName = "Movies";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        queryIndex(ddb, tableName);
        ddb.close();
    }

    public static void queryIndex(DynamoDbClient ddb, String tableName) {
        try {
            Map<String, String> expressionAttributesNames = new HashMap<>();
            expressionAttributesNames.put("#year", "year");
            Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
            expressionAttributeValues.put(":yearValue",
 AttributeValue.builder().n("2013").build());

            QueryRequest request = QueryRequest.builder()
                    .tableName(tableName)
                    .indexName("year-index")
                    .keyConditionExpression("#year = :yearValue")
                    .expressionAttributeNames(expressionAttributesNames)
                    .expressionAttributeValues(expressionAttributeValues)
                    .build();

            System.out.println("=== Movie Titles ===");
```

```
            QueryResponse response = ddb.query(request);
            response.items()
                    .forEach(movie ->
 System.out.println(movie.get("title").s()));

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener información sobre la API, consulte QueryCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
```

```
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create DynamoDB document client
var docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: "2012-08-10" });

var params = {
  ExpressionAttributeValues: {
    ":s": 2,
    ":e": 9,
    ":topic": "PHRASE",
  },
  KeyConditionExpression: "Season = :s and Episode > :e",
  FilterExpression: "contains (Subtitle, :topic)",
  TableName: "EPISODES_TABLE",
```

```
};

docClient.query(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.Items);
  }
});
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun queryDynTable(
    tableNameVal: String,
    partitionKeyName: String,
    partitionKeyVal: String,
    partitionAlias: String,
): Int {
    val attrNameAlias = mutableMapOf<String, String>()
    attrNameAlias[partitionAlias] = partitionKeyName

    // Set up mapping of the partition name with the value.
    val attrValues = mutableMapOf<String, AttributeValue>()
    attrValues[":$partitionKeyName"] = AttributeValue.S(partitionKeyVal)

    val request =
        QueryRequest {
```

```
            tableName = tableNameVal
            keyConditionExpression = "$partitionAlias = :$partitionKeyName"
            expressionAttributeNames = attrNameAlias
            this.expressionAttributeValues = attrValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.query(request)
        return response.count
    }
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);

    public function query(string $tableName, $key)
    {
        $expressionAttributeValues = [];
        $expressionAttributeNames = [];
        $keyConditionExpression = "";
        $index = 1;
```

```
        foreach ($key as $name => $value) {
            $keyConditionExpression .= "#" . array_key_first($value) . " = :v
$index,";
            $expressionAttributeNames["#" . array_key_first($value)] =
 array_key_first($value);
            $hold = array_pop($value);
            $expressionAttributeValues[":v$index"] = [
                array_key_first($hold) => array_pop($hold),
            ];
        }
        $keyConditionExpression = substr($keyConditionExpression, 0, -1);
        $query = [
            'ExpressionAttributeValues' => $expressionAttributeValues,
            'ExpressionAttributeNames' => $expressionAttributeNames,
            'KeyConditionExpression' => $keyConditionExpression,
            'TableName' => $tableName,
        ];
        return $this->dynamoDbClient->query($query);
    }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para PHP.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: invoca una consulta que devuelve elementos de DynamoDB con los valores de SongTitle y Artist especificados.

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

Salida:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Para obtener detalles de la API, consulte Query en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: invoca una consulta que devuelve elementos de DynamoDB con los valores de SongTitle y Artist especificados.

```
$invokeDDBQuery = @{
    TableName = 'Music'
    KeyConditionExpression = ' SongTitle = :SongTitle and Artist = :Artist'
    ExpressionAttributeValues = @{
        ':SongTitle' = 'Somewhere Down The Road'
        ':Artist' = 'No One You Know'
    } | ConvertTo-DDBItem
}
Invoke-DDBQuery @invokeDDBQuery | ConvertFrom-DDBItem
```

Salida:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Para obtener detalles de la API, consulte Query en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Consultar los elementos mediante una expresión de condición de clave.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None
```

```
    def query_movies(self, year):
        """
        Queries for movies that were released in the specified year.

        :param year: The year to query.
        :return: The list of movies that were released in the specified year.
        """
        try:
            response =
 self.table.query(KeyConditionExpression=Key("year").eq(year))
        except ClientError as err:
            logger.error(
                "Couldn't query for movies released in %s. Here's why: %s: %s",
                year,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Items"]
```

Consultar los elementos y proyectarlos para devolver un subconjunto de datos.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def query_and_project_movies(self, year, title_bounds):
        """
        Query for movies that were released in a specified year and that have
 titles
        that start within a range of letters. A projection expression is used
        to return a subset of data for each movie.

        :param year: The release year to query.
        :param title_bounds: The range of starting letters to query.
        :return: The list of movies.
        """
```

```
        try:
            response = self.table.query(
                ProjectionExpression="#yr, title, info.genres, info.actors[0]",
                ExpressionAttributeNames={"#yr": "year"},
                KeyConditionExpression=(
                    Key("year").eq(year)
                    & Key("title").between(
                        title_bounds["first"], title_bounds["second"]
                    )
                ),
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ValidationException":
                logger.warning(
                    "There's a validation error. Here's the message: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            else:
                logger.error(
                    "Couldn't query for movies. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return response["Items"]
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Queries for movies that were released in the specified year.
  #
  # @param year [Integer] The year to query.
  # @return [Array] The list of movies that were released in the specified year.
  def query_items(year)
    response = @table.query(
      key_condition_expression: '#yr = :year',
      expression_attribute_names: { '#yr' => 'year' },
      expression_attribute_values: { ':year' => year }
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't query for movies released in #{year}. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.items
  end
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Buscar las películas realizadas en el año especificado.

```
pub async fn movies_in_year(
    client: &Client,
    table_name: &str,
    year: u16,
) -> Result<Vec<Movie>, MovieError> {
    let results = client
        .query()
        .table_name(table_name)
        .key_condition_expression("#yr = :yyyy")
        .expression_attribute_names("#yr", "year")
        .expression_attribute_values(":yyyy",
 AttributeValue::N(year.to_string()))
        .send()
        .await?;

    if let Some(items) = results.items {
        let movies = items.iter().map(|v| v.into()).collect();
        Ok(movies)
    } else {
        Ok(vec![])
    }
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    TRY.
```

```
        " Query movies for a given year .
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_year }| ) ) ).
        DATA(lt_key_conditions) = VALUE /aws1/cl_dyncondition=>tt_keyconditions(
          ( VALUE /aws1/cl_dyncondition=>ts_keyconditions_maprow(
          key = 'year'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |EQ|
          ) ) ) ).
        oo_result = lo_dyn->query(
          iv_tablename = iv_table_name
          it_keyconditions = lt_key_conditions ).
        DATA(lt_items) = oo_result->get_items( ).
        "You can loop over the results to get item attributes.
        LOOP AT lt_items INTO DATA(lt_item).
          DATA(lo_title) = lt_item[ key = 'title' ]-value.
          DATA(lo_year) = lt_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_result->get_count( ).
        MESSAGE 'Item count is: ' && lv_count TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Para obtener información sobre la API, consulte Query en la Referencia de la API del AWSSDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB
```

```
    /// Get all the movies released in the specified year.
    ///
    /// - Parameter year: The release year of the movies to return.
    ///
    /// - Returns: An array of `Movie` objects describing each matching movie.
    ///
    func getMovies(fromYear year: Int) async throws -> [Movie] {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = QueryInput(
                expressionAttributeNames: [
                    "#y": "year"
                ],
                expressionAttributeValues: [
                    ":y": .n(String(year))
                ],
                keyConditionExpression: "#y = :y",
                tableName: self.tableName
            )
            // Use "Paginated" to get all the movies.
            // This lets the SDK handle the 'lastEvaluatedKey' property in
"QueryOutput".

            let pages = client.queryPaginated(input: input)

            var movieList: [Movie] = []
            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
                    continue
                }

                // Convert the found movies into `Movie` objects and return an
array
                // of them.

                for item in items {
                    let movie = try Movie(withItem: item)
                    movieList.append(movie)
```

```
                }
            }
            return movieList
        } catch {
            print("ERROR: getMovies:", dump(error))
            throw error
        }
    }
```

- Para obtener detalles de la API, consulte Query en la referencia de la API del SDK de AWS para Swift.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de **Scan** con un SDK de AWS o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `Scan`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Conceptos básicos
- Aceleración de lecturas con DAX
- Comparación de varios valores con un solo atributo

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</
returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string,
 AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0],
 info.directors, info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the
 LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
```

```
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
            throw;
        }
    }
```

- Para obtener información sobre la API, consulte Scan en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
##############################################################################
```

```
# function dynamodb_scan
#
# This function scans a DynamoDB table.
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -f filter_expression  -- The filter expression.
#       -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names.
#       -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values.
#       [-p projection_expression]  -- Optional projection expression.
#
#  Returns:
#       The items as json output.
#  And:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_scan() {
  local table_name filter_expression expression_attribute_names
 expression_attribute_values projection_expression response
  local option OPTARG # Required to use getopts command in a function.

  # ######################################
  # Function usage explanation
  ######################################
  function usage() {
    echo "function dynamodb_scan"
    echo "Scan a DynamoDB table."
    echo " -n table_name  -- The name of the table."
    echo " -f filter_expression  -- The filter expression."
    echo " -a expression_attribute_names -- Path to JSON file containing the
 expression attribute names."
    echo " -v expression_attribute_values -- Path to JSON file containing the
 expression attribute values."
    echo " [-p projection_expression]  -- Optional projection expression."
    echo ""
  }

  while getopts "n:f:a:v:p:h" option; do
    case "${option}" in
      n) table_name="${OPTARG}" ;;
      f) filter_expression="${OPTARG}" ;;
```

```
      a) expression_attribute_names="${OPTARG}" ;;
      v) expression_attribute_values="${OPTARG}" ;;
      p) projection_expression="${OPTARG}" ;;
      h)
        usage
        return 0
        ;;
      \?)
        echo "Invalid parameter"
        usage
        return 1
        ;;
    esac
  done
  export OPTIND=1

  if [[ -z "$table_name" ]]; then
    errecho "ERROR: You must provide a table name with the -n parameter."
    usage
    return 1
  fi

  if [[ -z "$filter_expression" ]]; then
    errecho "ERROR: You must provide a filter expression with the -f parameter."
    usage
    return 1
  fi

  if [[ -z "$expression_attribute_names" ]]; then
    errecho "ERROR: You must provide expression attribute names with the -a
parameter."
    usage
    return 1
  fi

  if [[ -z "$expression_attribute_values" ]]; then
    errecho "ERROR: You must provide expression attribute values with the -v
parameter."
    usage
    return 1
  fi

  if [[ -z "$projection_expression" ]]; then
    response=$(aws dynamodb scan \
```

```
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values")
  else
    response=$(aws dynamodb scan \
      --table-name "$table_name" \
      --filter-expression "$filter_expression" \
      --expression-attribute-names file://"$expression_attribute_names" \
      --expression-attribute-values file://"$expression_attribute_values" \
      --projection-expression "$projection_expression")
  fi

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports scan operation failed.$response"
    return 1
  fi

  echo "$response"

  return 0
}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
```

```
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
################################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información sobre la API, consulte Scan en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Scan an Amazon DynamoDB table.
/*!
  \sa scanTable()
  \param tableName: Name for the DynamoDB table.
  \param projectionExpression: An optional projection expression, ignored if
 empty.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */

bool AwsDoc::DynamoDB::scanTable(const Aws::String &tableName,
                                 const Aws::String &projectionExpression,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);
    Aws::DynamoDB::Model::ScanRequest request;
    request.SetTableName(tableName);

    if (!projectionExpression.empty())
        request.SetProjectionExpression(projectionExpression);

    Aws::Vector<Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>>
 all_items;
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 last_evaluated_key; // Used for pagination;
    do {
        if (!last_evaluated_key.empty()) {
            request.SetExclusiveStartKey(last_evaluated_key);
        }
        const Aws::DynamoDB::Model::ScanOutcome &outcome =
 dynamoClient.Scan(request);
        if (outcome.IsSuccess()) {
```

```
            // Reference the retrieved items.
            const Aws::Vector<Aws::Map<Aws::String,
 Aws::DynamoDB::Model::AttributeValue>> &items = outcome.GetResult().GetItems();
            all_items.insert(all_items.end(), items.begin(), items.end());

            last_evaluated_key = outcome.GetResult().GetLastEvaluatedKey();
        }
        else {
            std::cerr << "Failed to Scan items: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }

    } while (!last_evaluated_key.empty());

    if (!all_items.empty()) {
        std::cout << "Number of items retrieved from scan: " << all_items.size()
                  << std::endl;
        // Iterate each item and print.
        for (const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 &itemMap: all_items) {
            std::cout << "*************************************************"
                      << std::endl;
            // Output each retrieved field and its value.
            for (const auto &itemEntry: itemMap)
                std::cout << itemEntry.first << ": " << itemEntry.second.GetS()
                          << std::endl;
        }
    }

    else {
        std::cout << "No items found in table: " << tableName << std::endl;
    }

    return true;
}
```

- Para obtener información sobre la API, consulte Scan en la referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Análisis de una tabla

En el siguiente ejemplo de `scan` se escanea toda la tabla `MusicCollection` y, a continuación, se reducen los resultados a las canciones del artista “No One You Know”. Para cada elemento, solo se devuelven el título del álbum y el título de la canción.

```
aws dynamodb scan \
    --table-name MusicCollection \
    --filter-expression "Artist = :a" \
    --projection-expression "#ST, #AT" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json
```

Contenido de `expression-attribute-names.json`:

```
{
    "#ST": "SongTitle",
    "#AT":"AlbumTitle"
}
```

Contenido de `expression-attribute-values.json`:

```
{
    ":a": {"S": "No One You Know"}
}
```

Salida:

```
{
    "Count": 2,
    "Items": [
        {
            "SongTitle": {
                "S": "Call Me Today"
            },
            "AlbumTitle": {
                "S": "Somewhat Famous"
            }
```

```
        },
        {
            "SongTitle": {
                "S": "Scared of My Shadow"
            },
            "AlbumTitle": {
                "S": "Blue Sky Blues"
            }
        }
    ],
    "ScannedCount": 3,
    "ConsumedCapacity": null
}
```

Para obtener más información, consulte Uso de operaciones de análisis en DynamoDB en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte Scan en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
```

```
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Scan gets all movies in the DynamoDB table that were released in a range of
 years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
// expressions.
func (basics TableBasics) Scan(ctx context.Context, startYear int, endYear int)
 ([]Movie, error) {
 var movies []Movie
 var err error
 var response *dynamodb.ScanOutput
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"),
 expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(ctx)
   if err != nil {
```

```
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why:
 %v\n",
     startYear, endYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

Defina una estructura Película para utilizar en este ejemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
```

```
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obtener información sobre la API, consulte Scan en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Escanea una tabla de Amazon DynamoDB con DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import java.util.Map;
import java.util.Set;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * To scan items from an Amazon DynamoDB table using the AWS SDK for Java V2,
 * its better practice to use the
 * Enhanced Client, See the EnhancedScanRecords example.
 */

public class DynamoDBScanItems {
    public static void main(String[] args) {

        final String usage = """

                Usage:
                    <tableName>

                Where:
                    tableName - The Amazon DynamoDB table to get information from
 (for example, Music3).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        Region region = Region.US_EAST_1;
```

```
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        scanItems(ddb, tableName);
        ddb.close();
    }

    public static void scanItems(DynamoDbClient ddb, String tableName) {
        try {
            ScanRequest scanRequest = ScanRequest.builder()
                    .tableName(tableName)
                    .build();

            ScanResponse response = ddb.scan(scanRequest);
            for (Map<String, AttributeValue> item : response.items()) {
                Set<String> keys = item.keySet();
                for (String key : keys) {
                    System.out.println("The key name is " + key + "\n");
                    System.out.println("The value is " + item.get(key).s());
                }
            }

        } catch (DynamoDbException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

- Para obtener información sobre la API, consulte Scan en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener información sobre la API, consulte ScanCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, ScanCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ScanCommand({
    ProjectionExpression: "#Name, Color, AvgLifeSpan",
    ExpressionAttributeNames: { "#Name": "Name" },
    TableName: "Birds",
  });

  const response = await docClient.send(command);
  for (const bird of response.Items) {
    console.log(`${bird.Name} - (${bird.Color}, ${bird.AvgLifeSpan})`);
  }
  return response;
};
```

- Para obtener información sobre la API, consulte Scan en la referencia de la API de AWS SDK para JavaScript.

SDK para JavaScript (v2)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
// Load the AWS SDK for Node.js.
var AWS = require("aws-sdk");
// Set the AWS Region.
AWS.config.update({ region: "REGION" });

// Create DynamoDB service object.
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });

const params = {
  // Specify which items in the results are returned.
  FilterExpression: "Subtitle = :topic AND Season = :s AND Episode = :e",
  // Define the expression attribute value, which are substitutes for the values
 you want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: "SubTitle2" },
    ":s": { N: 1 },
    ":e": { N: 2 },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "Season, Episode, Title, Subtitle",
  TableName: "EPISODES_TABLE",
};

ddb.scan(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
    data.Items.forEach(function (element, index, array) {
      console.log(
        "printing",
        element.Title.S + " (" + element.Subtitle.S + ")"
      );
    });
```

```
  }
});
```

- Para obtener información, consulte la Guía para desarrolladores de AWS SDK para JavaScript.
- Para obtener información sobre la API, consulte Scan en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun scanItems(tableNameVal: String) {
    val request =
        ScanRequest {
            tableName = tableNameVal
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        val response = ddb.scan(request)
        response.items?.forEach { item ->
            item.keys.forEach { key ->
                println("The key name is $key\n")
                println("The value is ${item[key]}")
            }
        }
    }
}
```

- Para obtener información sobre la API, consulte Scan en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

    public function scan(string $tableName, array $key, string $filters)
    {
        $query = [
            'ExpressionAttributeNames' => ['#year' => 'year'],
            'ExpressionAttributeValues' => [
                ":min" => ['N' => '1990'],
                ":max" => ['N' => '1999'],
            ],
            'FilterExpression' => "#year between :min and :max",
            'TableName' => $tableName,
        ];
        return $this->dynamoDbClient->scan($query);
    }
```

- Para obtener información sobre la API, consulte Scan en la referencia de la API de AWS SDK para PHP.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: devuelve todos los elementos de la tabla Music.

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

Salida:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
Genre                          Country
Artist                         No One You Know
Price                          1.98
CriticRating                   8.4
SongTitle                      My Dog Spot
AlbumTitle                     Hey Now
```

Ejemplo 2: devuelve los elementos de la tabla Music con un CriticRating superior o igual a nueve.

```
$scanFilter = @{
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
        }
    }
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

Salida:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Para obtener detalles de la API, consulte Scan en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: devuelve todos los elementos de la tabla Music.

```
Invoke-DDBScan -TableName 'Music' | ConvertFrom-DDBItem
```

Salida:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
Genre                          Country
Artist                         No One You Know
Price                          1.98
CriticRating                   8.4
SongTitle                      My Dog Spot
AlbumTitle                     Hey Now
```

Ejemplo 2: devuelve los elementos de la tabla Music con un CriticRating superior o igual a nueve.

```
$scanFilter = @{
        CriticRating = [Amazon.DynamoDBv2.Model.Condition]@{
            AttributeValueList = @(@{N = '9'})
            ComparisonOperator = 'GE'
```

```
        }
    }
    Invoke-DDBScan -TableName 'Music' -ScanFilter $scanFilter | ConvertFrom-
DDBItem
```

Salida:

```
Name                           Value
----                           -----
Genre                          Country
Artist                         No One You Know
Price                          1.94
CriticRating                   9
SongTitle                      Somewhere Down The Road
AlbumTitle                     Somewhat Famous
```

- Para obtener detalles de la API, consulte Scan en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
```

```
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def scan_movies(self, year_range):
        """
        Scans for movies that were released in a range of years.
        Uses a projection expression to return a subset of data for each movie.

        :param year_range: The range of years to retrieve.
        :return: The list of movies released in the specified years.
        """
        movies = []
        scan_kwargs = {
            "FilterExpression": Key("year").between(
                year_range["first"], year_range["second"]
            ),
            "ProjectionExpression": "#yr, title, info.rating",
            "ExpressionAttributeNames": {"#yr": "year"},
        }
        try:
            done = False
            start_key = None
            while not done:
                if start_key:
                    scan_kwargs["ExclusiveStartKey"] = start_key
                response = self.table.scan(**scan_kwargs)
```

```
                movies.extend(response.get("Items", []))
                start_key = response.get("LastEvaluatedKey", None)
                done = start_key is None
        except ClientError as err:
            logger.error(
                "Couldn't scan for movies. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise

        return movies
```

- Para obtener información sobre la API, consulte Scan en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Scans for movies that were released in a range of years.
  # Uses a projection expression to return a subset of data for each movie.
  #
  # @param year_range [Hash] The range of years to retrieve.
```

```
  # @return [Array] The list of movies released in the specified years.
  def scan_items(year_range)
    movies = []
    scan_hash = {
      filter_expression: '#yr between :start_yr and :end_yr',
      projection_expression: '#yr, title, info.rating',
      expression_attribute_names: { '#yr' => 'year' },
      expression_attribute_values: {
        ':start_yr' => year_range[:start], ':end_yr' => year_range[:end]
      }
    }
    done = false
    start_key = nil
    until done
      scan_hash[:exclusive_start_key] = start_key unless start_key.nil?
      response = @table.scan(scan_hash)
      movies.concat(response.items) unless response.items.empty?
      start_key = response.last_evaluated_key
      done = start_key.nil?
    end
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't scan for movies. Here's why:")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    movies
  end
```

- Para obtener información sobre la API, consulte Scan en la referencia de la API de AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
pub async fn list_items(client: &Client, table: &str, page_size: Option<i32>) ->
 Result<(), Error> {
    let page_size = page_size.unwrap_or(10);
    let items: Result<Vec<_>, _> = client
        .scan()
        .table_name(table)
        .limit(page_size)
        .into_paginator()
        .items()
        .send()
        .collect()
        .await;

    println!("Items in table (up to {page_size}):");
    for item in items? {
        println!("   {:?}", item);
    }

    Ok(())
}
```

- Para obtener información sobre la API, consulte Scan en la referencia de la API de AWS SDK para Rust.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    TRY.
        " Scan movies for rating greater than or equal to the rating specified
        DATA(lt_attributelist) = VALUE /aws1/
cl_dynattributevalue=>tt_attributevaluelist(
            ( NEW /aws1/cl_dynattributevalue( iv_n = |{ iv_rating }| ) ) ).
```

```
        DATA(lt_filter_conditions) = VALUE /aws1/
cl_dyncondition=>tt_filterconditionmap(
          ( VALUE /aws1/cl_dyncondition=>ts_filterconditionmap_maprow(
          key = 'rating'
          value = NEW /aws1/cl_dyncondition(
          it_attributevaluelist = lt_attributelist
          iv_comparisonoperator = |GE|
          ) ) ) ).
        oo_scan_result = lo_dyn->scan( iv_tablename = iv_table_name
          it_scanfilter = lt_filter_conditions ).
        DATA(lt_items) = oo_scan_result->get_items( ).
        LOOP AT lt_items INTO DATA(lo_item).
          " You can loop over to get individual attributes.
          DATA(lo_title) = lo_item[ key = 'title' ]-value.
          DATA(lo_year) = lo_item[ key = 'year' ]-value.
        ENDLOOP.
        DATA(lv_count) = oo_scan_result->get_count( ).
        MESSAGE 'Found ' && lv_count && ' items' TYPE 'I'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
    ENDTRY.
```

- Para obtener información sobre la API, consulte Scan en la Referencia de la API del AWSSDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB


    /// Return an array of `Movie` objects released in the specified range of
    /// years.
```

```
    ///
    /// - Parameters:
    ///   - firstYear: The first year of movies to return.
    ///   - lastYear: The last year of movies to return.
    ///   - startKey: A starting point to resume processing; always use `nil`.
    ///
    /// - Returns: An array of `Movie` objects describing the matching movies.
    ///
    /// > Note: The `startKey` parameter is used by this function when
    ///   recursively calling itself, and should always be `nil` when calling
    ///   directly.
    ///
    func getMovies(firstYear: Int, lastYear: Int,
                   startKey: [Swift.String: DynamoDBClientTypes.AttributeValue]?
= nil)
        async throws -> [Movie]
    {
        do {
            var movieList: [Movie] = []

            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            let input = ScanInput(
                consistentRead: true,
                exclusiveStartKey: startKey,
                expressionAttributeNames: [
                    "#y": "year" // `year` is a reserved word, so use `#y`
instead.
                ],
                expressionAttributeValues: [
                    ":y1": .n(String(firstYear)),
                    ":y2": .n(String(lastYear))
                ],
                filterExpression: "#y BETWEEN :y1 AND :y2",
                tableName: self.tableName
            )

            let pages = client.scanPaginated(input: input)

            for try await page in pages {
                guard let items = page.items else {
                    print("Error: no items returned.")
```

```
                continue
            }

            // Build an array of `Movie` objects for the returned items.

            for item in items {
                let movie = try Movie(withItem: item)
                movieList.append(movie)
            }
        }
        return movieList

    } catch {
        print("ERROR: getMovies with scan:", dump(error))
        throw error
    }
}
```

- Para obtener detalles de la API, consulte Scan en la referencia de la API del SDK de AWS para Swift.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de **`UpdateItem`** con un SDK de AWS o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `UpdateItem`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en los siguientes ejemplos de código:

- Conceptos básicos
- Actualización condicional del TTL de un elemento
- Recuento de operadores de expresión
- Creación y administración de tablas globales de MRSC
- Realización de operaciones de lista
- Realización de operaciones de asignación

- Realización de operaciones de conjunto
- Descripción del orden de las expresiones de actualización
- Actualización del TTL de un elemento
- Uso de operaciones de contador atómico
- Uso de operaciones condicionales
- Uso de nombres de atributo de expresión

.NET

SDK para .NET (v4)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
```

```
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
```

```
        }
    }
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para .NET.

Bash

AWS CLI con Bash script

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
###############################################################################
# function dynamodb_update_item
#
# This function updates an item in a DynamoDB table.
#
#
# Parameters:
#       -n table_name  -- The name of the table.
#       -k keys  -- Path to json file containing the keys that identify the item
 to update.
#       -e update expression  -- An expression that defines one or more
 attributes to be updated.
#       -v values  -- Path to json file containing the update values.
#
#  Returns:
#       0 - If successful.
#       1 - If it fails.
###############################################################################
function dynamodb_update_item() {
  local table_name keys update_expression values response
  local option OPTARG # Required to use getopts command in a function.

  ######################################
```

```
 # Function usage explanation
 ####################################
 function usage() {
   echo "function dynamodb_update_item"
   echo "Update an item in a DynamoDB table."
   echo " -n table_name  -- The name of the table."
   echo " -k keys  -- Path to json file containing the keys that identify the
item to update."
   echo " -e update expression  -- An expression that defines one or more
attributes to be updated."
   echo " -v values  -- Path to json file containing the update values."
   echo ""
 }

 while getopts "n:k:e:v:h" option; do
   case "${option}" in
     n) table_name="${OPTARG}" ;;
     k) keys="${OPTARG}" ;;
     e) update_expression="${OPTARG}" ;;
     v) values="${OPTARG}" ;;
     h)
       usage
       return 0
       ;;
     \?)
       echo "Invalid parameter"
       usage
       return 1
       ;;
   esac
 done
 export OPTIND=1

 if [[ -z "$table_name" ]]; then
   errecho "ERROR: You must provide a table name with the -n parameter."
   usage
   return 1
 fi

 if [[ -z "$keys" ]]; then
   errecho "ERROR: You must provide a keys json file path the -k parameter."
   usage
   return 1
 fi
```

```
  if [[ -z "$update_expression" ]]; then
    errecho "ERROR: You must provide an update expression with the -e parameter."
    usage
    return 1
  fi

  if [[ -z "$values" ]]; then
    errecho "ERROR: You must provide a values json file path the -v parameter."
    usage
    return 1
  fi

  iecho "Parameters:\n"
  iecho "    table_name:   $table_name"
  iecho "    keys:   $keys"
  iecho "    update_expression:   $update_expression"
  iecho "    values:   $values"

  response=$(aws dynamodb update-item \
    --table-name "$table_name" \
    --key file://"$keys" \
    --update-expression "$update_expression" \
    --expression-attribute-values file://"$values")

  local error_code=${?}

  if [[ $error_code -ne 0 ]]; then
    aws_cli_error_log $error_code
    errecho "ERROR: AWS reports update-item operation failed.$response"
    return 1
  fi

  return 0

}
```

Las funciones de utilidad utilizadas en este ejemplo.

```
###############################################################################
# function iecho
#
# This function enables the script to display the specified text only if
```

```
# the global variable $VERBOSE is set to true.
###############################################################################
function iecho() {
  if [[ $VERBOSE == true ]]; then
    echo "$@"
  fi
}

###############################################################################
# function errecho
#
# This function outputs everything sent to it to STDERR (standard error output).
###############################################################################
function errecho() {
  printf "%s\n" "$*" 1>&2
}

###############################################################################
# function aws_cli_error_log()
#
# This function is used to log the error messages from the AWS CLI.
#
# See https://docs.aws.amazon.com/cli/latest/topic/return-codes.html#cli-aws-
help-return-codes.
#
# The function expects the following argument:
#         $1 - The error code returned by the AWS CLI.
#
#  Returns:
#          0: - Success.
#
###############################################################################
function aws_cli_error_log() {
  local err_code=$1
  errecho "Error code : $err_code"
  if [ "$err_code" == 1 ]; then
    errecho "  One or more S3 transfers failed."
  elif [ "$err_code" == 2 ]; then
    errecho "  Command line failed to parse."
  elif [ "$err_code" == 130 ]; then
    errecho "  Process received SIGINT."
  elif [ "$err_code" == 252 ]; then
    errecho "  Command syntax invalid."
  elif [ "$err_code" == 253 ]; then
```

```
    errecho "  The system environment or configuration was invalid."
  elif [ "$err_code" == 254 ]; then
    errecho "  The service returned an error."
  elif [ "$err_code" == 255 ]; then
    errecho "  255 is a catch-all error."
  fi

  return 0
}
```

- Para obtener información sobre la API, consulte UpdateItem en la Referencia de comandos de la AWS CLI.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Update an Amazon DynamoDB table item.
/*!
  \sa updateItem()
  \param tableName: The table name.
  \param partitionKey: The partition key.
  \param partitionValue: The value for the partition key.
  \param attributeKey: The key for the attribute to be updated.
  \param attributeValue: The value for the attribute to be updated.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
  */

/*
 *  The example code only sets/updates an attribute value. It processes
 *  the attribute value as a string, even if the value could be interpreted
 *  as a number. Also, the example code does not remove an existing attribute
 *  from the key value.
```

```
 */

bool AwsDoc::DynamoDB::updateItem(const Aws::String &tableName,
                                  const Aws::String &partitionKey,
                                  const Aws::String &partitionValue,
                                  const Aws::String &attributeKey,
                                  const Aws::String &attributeValue,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // *** Define UpdateItem request arguments.
    // Define TableName argument.
    Aws::DynamoDB::Model::UpdateItemRequest request;
    request.SetTableName(tableName);

    // Define KeyName argument.
    Aws::DynamoDB::Model::AttributeValue attribValue;
    attribValue.SetS(partitionValue);
    request.AddKey(partitionKey, attribValue);

    // Construct the SET update expression argument.
    Aws::String update_expression("SET #a = :valueA");
    request.SetUpdateExpression(update_expression);

    // Construct attribute name argument.
    Aws::Map<Aws::String, Aws::String> expressionAttributeNames;
    expressionAttributeNames["#a"] = attributeKey;
    request.SetExpressionAttributeNames(expressionAttributeNames);

    // Construct attribute value argument.
    Aws::DynamoDB::Model::AttributeValue attributeUpdatedValue;
    attributeUpdatedValue.SetS(attributeValue);
    Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
 expressionAttributeValues;
    expressionAttributeValues[":valueA"] = attributeUpdatedValue;
    request.SetExpressionAttributeValues(expressionAttributeValues);

    // Update the item.
    const Aws::DynamoDB::Model::UpdateItemOutcome &outcome =
 dynamoClient.UpdateItem(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Item was updated" << std::endl;
```

```
    } else {
        std::cerr << outcome.GetError().GetMessage() << std::endl;
        return false;
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Código que espera a que se active la tabla.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
```

```
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Ejemplo 1: Actualización de un elemento de una tabla

En el siguiente ejemplo de `update-item`, se actualiza un elemento de la tabla `MusicCollection`. Añade un nuevo atributo (`Year`) y modifica el atributo `AlbumTitle`. En la respuesta se muestran todos los atributos del elemento, tal como aparecen después de la actualización.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --return-values ALL_NEW \
    --return-consumed-capacity TOTAL \
    --return-item-collection-metrics SIZE
```

Contenido de `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Contenido de `expression-attribute-names.json`:

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
}
```

Contenido de `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

Salida:

```
{
    "Attributes": {
        "AlbumTitle": {
            "S": "Louder Than Ever"
        },
        "Awards": {
            "N": "10"
        },
        "Artist": {
            "S": "Acme Band"
        },
        "Year": {
            "N": "2015"
        },
        "SongTitle": {
            "S": "Happy Day"
        }
    },
    "ConsumedCapacity": {
        "TableName": "MusicCollection",
        "CapacityUnits": 3.0
    },
    "ItemCollectionMetrics": {
        "ItemCollectionKey": {
            "Artist": {
                "S": "Acme Band"
            }
        },
```

```
        "SizeEstimateRangeGB": [
            0.0,
            1.0
        ]
    }
}
```

Para obtener más información, consulte Escritura de un elemento en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 2: actualización de un elemento de forma condicional

En el siguiente ejemplo se actualiza un elemento de la tabla `MusicCollection`, pero solo si el elemento existente aún no tiene un atributo `Year`.

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --condition-expression "attribute_not_exists(#Y)"
```

Contenido de `key.json`:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Contenido de `expression-attribute-names.json`:

```
{
    "#Y":"Year",
    "#AT":"AlbumTitle"
}
```

Contenido de `expression-attribute-values.json`:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
```

```
}
```

Si el elemento ya tiene un atributo `Year`, DynamoDB devuelve el siguiente resultado.

```
An error occurred (ConditionalCheckFailedException) when calling the UpdateItem
 operation: The conditional request failed
```

Para obtener más información, consulte Escritura de un elemento en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte UpdateItem en la Referencia de comandos de la AWS CLI.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import (
 "context"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified
 table.
```

```
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the
 update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx,
 &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}
```

Defina una estructura Película para utilizar en este ejemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}
```

```
// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Actualiza un elemento de una tabla mediante DynamoDbClient.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.AttributeAction;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.AttributeValueUpdate;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import java.util.HashMap;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
```

```
 * To update an Amazon DynamoDB table using the AWS SDK for Java V2, its better
 * practice to use the
 * Enhanced Client, See the EnhancedModifyItem example.
 */
public class UpdateItem {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableName> <key> <keyVal> <name> <updateVal>

                Where:
                    tableName - The Amazon DynamoDB table (for example, Music3).
                    key - The name of the key in the table (for example, Artist).
                    keyVal - The value of the key (for example, Famous Band).
                    name - The name of the column where the value is updated (for
 example, Awards).
                    updateVal - The value used to update an item (for example,
 14).
                 Example:
                    UpdateItem Music3 Artist Famous Band Awards 14
                """;

        if (args.length != 5) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableName = args[0];
        String key = args[1];
        String keyVal = args[2];
        String name = args[3];
        String updateVal = args[4];

        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();
        updateTableItem(ddb, tableName, key, keyVal, name, updateVal);
        ddb.close();
    }

    public static void updateTableItem(DynamoDbClient ddb,
            String tableName,
```

```
            String key,
            String keyVal,
            String name,
            String updateVal) {

        HashMap<String, AttributeValue> itemKey = new HashMap<>();
        itemKey.put(key, AttributeValue.builder()
                .s(keyVal)
                .build());

        HashMap<String, AttributeValueUpdate> updatedValues = new HashMap<>();
        updatedValues.put(name, AttributeValueUpdate.builder()
                .value(AttributeValue.builder().s(updateVal).build())
                .action(AttributeAction.PUT)
                .build());

        UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(itemKey)
                .attributeUpdates(updatedValues)
                .build();

        try {
            ddb.updateItem(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("The Amazon DynamoDB table was updated!");
    }
}
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Este ejemplo utiliza el cliente de documentos para simplificar el trabajo con elementos en DynamoDB. Para obtener información sobre la API, consulte UpdateCommand.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun updateTableItem(
    tableNameVal: String,
    keyName: String,
    keyVal: String,
    name: String,
    updateVal: String,
) {
    val itemKey = mutableMapOf<String, AttributeValue>()
    itemKey[keyName] = AttributeValue.S(keyVal)

    val updatedValues = mutableMapOf<String, AttributeValueUpdate>()
    updatedValues[name] =
        AttributeValueUpdate {
            value = AttributeValue.S(updateVal)
            action = AttributeAction.Put
        }

    val request =
        UpdateItemRequest {
            tableName = tableNameVal
            key = itemKey
            attributeUpdates = updatedValues
        }

    DynamoDbClient.fromEnvironment { region = "us-east-1" }.use { ddb ->
        ddb.updateItem(request)
        println("Item in $tableNameVal was updated")
    }
}
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        echo "What rating would you like to give {$movie['Item']['title']['S']}?
\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $service->updateItemAttributeByKey($tableName, $key, 'rating', 'N',
 $rating);

    public function updateItemAttributeByKey(
        string $tableName,
        array $key,
        string $attributeName,
        string $attributeType,
        string $newValue
    ) {
        $this->dynamoDbClient->updateItem([
            'Key' => $key['Item'],
            'TableName' => $tableName,
            'UpdateExpression' => "set #NV=:NV",
            'ExpressionAttributeNames' => [
                '#NV' => $attributeName,
            ],
            'ExpressionAttributeValues' => [
                ':NV' => [
                    $attributeType => $newValue
                ]
```

```
            ],
        ]);
    }
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para PHP.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: establece el atributo de género como Rap en el elemento de DynamoDB con la clave de partición SongTitle y la clave de clasificación Artist.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

Salida:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Para obtener detalles de la API, consulte UpdateItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: establece el atributo de género como Rap en el elemento de DynamoDB con la clave de partición SongTitle y la clave de clasificación Artist.

```
$key = @{
    SongTitle = 'Somewhere Down The Road'
    Artist = 'No One You Know'
} | ConvertTo-DDBItem

$updateDdbItem = @{
    TableName = 'Music'
    Key = $key
    UpdateExpression = 'set Genre = :val1'
    ExpressionAttributeValue = (@{
        ':val1' = ([Amazon.DynamoDBv2.Model.AttributeValue]'Rap')
    })
}
Update-DDBItem @updateDdbItem
```

Salida:

```
Name                           Value
----                           -----
Genre                          Rap
```

- Para obtener detalles de la API, consulte UpdateItem en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Actualizar un elemento con una expresión de actualización.

```
class Movies:
    """Encapsulates an Amazon DynamoDB table of movie data.

    Example data structure for a movie record in this table:
        {
            "year": 1999,
            "title": "For Love of the Game",
            "info": {
                "directors": ["Sam Raimi"],
                "release_date": "1999-09-15T00:00:00Z",
                "rating": 6.3,
                "plot": "A washed up pitcher flashes through his career.",
                "rank": 4987,
                "running_time_secs": 8220,
                "actors": [
                    "Kevin Costner",
                    "Kelly Preston",
                    "John C. Reilly"
                ]
            }
        }
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource
        # The table variable is set during the scenario in the call to
        # 'exists' if the table exists. Otherwise, it is set by 'create_table'.
        self.table = None


    def update_movie(self, title, year, rating, plot):
        """
        Updates rating and plot data for a movie in the table.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating: The updated rating to the give the movie.
        :param plot: The updated plot summary to give the movie.
        :return: The fields that were updated, with their new values.
        """
```

```
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating=:r, info.plot=:p",
                ExpressionAttributeValues={":r": Decimal(str(rating)), ":p":
 plot},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

Actualizar un elemento con una expresión de actualización que incluye una operación aritmética.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def update_rating(self, title, year, rating_change):
        """
        Updates the quality rating of a movie in the table by using an arithmetic
        operation in the update expression. By specifying an arithmetic
 operation,
        you can adjust a value in a single request, rather than first getting its
        value and then setting its new value.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param rating_change: The amount to add to the current rating for the
 movie.
        :return: The updated rating.
```

```
        """
        try:
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="set info.rating = info.rating + :val",
                ExpressionAttributeValues={":val": Decimal(str(rating_change))},
                ReturnValues="UPDATED_NEW",
            )
        except ClientError as err:
            logger.error(
                "Couldn't update movie %s in table %s. Here's why: %s: %s",
                title,
                self.table.name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["Attributes"]
```

Actualizar un elemento solo cuando cumpla determinadas condiciones.

```
class UpdateQueryWrapper:
    def __init__(self, table):
        self.table = table


    def remove_actors(self, title, year, actor_threshold):
        """
        Removes an actor from a movie, but only when the number of actors is
 greater
        than a specified threshold. If the movie does not list more than the
 threshold,
        no actors are removed.

        :param title: The title of the movie to update.
        :param year: The release year of the movie to update.
        :param actor_threshold: The threshold of actors to check.
        :return: The movie data after the update.
        """
        try:
```

```
            response = self.table.update_item(
                Key={"year": year, "title": title},
                UpdateExpression="remove info.actors[0]",
                ConditionExpression="size(info.actors) > :num",
                ExpressionAttributeValues={":num": actor_threshold},
                ReturnValues="ALL_NEW",
            )
        except ClientError as err:
            if err.response["Error"]["Code"] ==
"ConditionalCheckFailedException":
                logger.warning(
                    "Didn't update %s because it has fewer than %s actors.",
                    title,
                    actor_threshold + 1,
                )
            else:
                logger.error(
                    "Couldn't update movie %s. Here's why: %s: %s",
                    title,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response["Attributes"]
```

- Para obtener información la API, consulte UpdateItem en la referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
class DynamoDBBasics
  attr_reader :dynamo_resource, :table

  def initialize(table_name)
    client = Aws::DynamoDB::Client.new(region: 'us-east-1')
    @dynamo_resource = Aws::DynamoDB::Resource.new(client: client)
    @table = @dynamo_resource.table(table_name)
  end

  # Updates rating and plot data for a movie in the table.
  #
  # @param movie [Hash] The title, year, plot, rating of the movie.
  def update_item(movie)
    response = @table.update_item(
      key: { 'year' => movie[:year], 'title' => movie[:title] },
      update_expression: 'set info.rating=:r',
      expression_attribute_values: { ':r' => movie[:rating] },
      return_values: 'UPDATED_NEW'
    )
  rescue Aws::DynamoDB::Errors::ServiceError => e
    puts("Couldn't update movie #{movie[:title]} (#{movie[:year]}) in table
 #{@table.name}\n")
    puts("\t#{e.code}: #{e.message}")
    raise
  else
    response.attributes
  end
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para Ruby.

SAP ABAP

SDK para SAP ABAP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
    TRY.
        oo_output = lo_dyn->updateitem(
          iv_tablename        = iv_table_name
          it_key              = it_item_key
          it_attributeupdates = it_attribute_updates ).
        MESSAGE '1 item updated in DynamoDB Table' && iv_table_name TYPE 'I'.
      CATCH /aws1/cx_dyncondalcheckfaile00.
        MESSAGE 'A condition specified in the operation could not be evaluated.'
TYPE 'E'.
      CATCH /aws1/cx_dynresourcenotfoundex.
        MESSAGE 'The table or index does not exist' TYPE 'E'.
      CATCH /aws1/cx_dyntransactconflictex.
        MESSAGE 'Another transaction is using the item' TYPE 'E'.
    ENDTRY.
```

- Para obtener información sobre la API, consulte UpdateItem en la Referencia de la API del AWSSDK para SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
import AWSDynamoDB


    /// Update the specified movie with new `rating` and `plot` information.
    ///
    /// - Parameters:
    ///   - title: The title of the movie to update.
    ///   - year: The release year of the movie to update.
    ///   - rating: The new rating for the movie.
    ///   - plot: The new plot summary string for the movie.
    ///
    /// - Returns: An array of mappings of attribute names to their new
```

```
    ///   listing each item actually changed. Items that didn't need to change
    ///   aren't included in this list. `nil` if no changes were made.
    ///
    func update(title: String, year: Int, rating: Double? = nil, plot: String? =
nil) async throws
        -> [Swift.String: DynamoDBClientTypes.AttributeValue]?
    {
        do {
            guard let client = self.ddbClient else {
                throw MoviesError.UninitializedClient
            }

            // Build the update expression and the list of expression attribute
            // values. Include only the information that's changed.

            var expressionParts: [String] = []
            var attrValues: [Swift.String: DynamoDBClientTypes.AttributeValue] =
[:]

            if rating != nil {
                expressionParts.append("info.rating=:r")
                attrValues[":r"] = .n(String(rating!))
            }
            if plot != nil {
                expressionParts.append("info.plot=:p")
                attrValues[":p"] = .s(plot!)
            }
            let expression = "set \(expressionParts.joined(separator: ", "))"

            let input = UpdateItemInput(
                // Create substitution tokens for the attribute values, to ensure
                // no conflicts in expression syntax.
                expressionAttributeValues: attrValues,
                // The key identifying the movie to update consists of the
release
                // year and title.
                key: [
                    "year": .n(String(year)),
                    "title": .s(title)
                ],
                returnValues: .updatedNew,
                tableName: self.tableName,
                updateExpression: expression
            )
```

```
            let output = try await client.updateItem(input: input)

            guard let attributes: [Swift.String:
DynamoDBClientTypes.AttributeValue] = output.attributes else {
                throw MoviesError.InvalidAttributes
            }
            return attributes
        } catch {
            print("ERROR: update:", dump(error))
            throw error
        }
    }
```

- Para obtener detalles de la API, consulte UpdateItem en la referencia de la API del SDK de AWS para Swift.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de **`UpdateTable`** con un SDK de AWS o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `UpdateTable`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en su contexto en los siguientes ejemplos de código:

- Creación y administración de tablas globales que muestren MREC
- Creación y administración de tablas globales de MRSC
- Administración de índices secundarios globales
- Actualización de la configuración de rendimiento en caliente de una tabla
- Trabajo con tablas globales y coherencia final de replicación de varias regiones (MREC)
- Trabajo con el cifrado de tablas

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
//! Update a DynamoDB table.
/*!
  \sa updateTable()
  \param tableName: Name for the DynamoDB table.
  \param readCapacity: Provisioned read capacity.
  \param writeCapacity: Provisioned write capacity.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::updateTable(const Aws::String &tableName,
                                   long long readCapacity, long long
 writeCapacity,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::cout << "Updating " << tableName << " with new provisioned throughput
 values"
              << std::endl;
    std::cout << "Read capacity : " << readCapacity << std::endl;
    std::cout << "Write capacity: " << writeCapacity << std::endl;

    Aws::DynamoDB::Model::UpdateTableRequest request;
    Aws::DynamoDB::Model::ProvisionedThroughput provisionedThroughput;

 provisionedThroughput.WithReadCapacityUnits(readCapacity).WithWriteCapacityUnits(
           writeCapacity);

 request.WithProvisionedThroughput(provisionedThroughput).WithTableName(tableName);

    const Aws::DynamoDB::Model::UpdateTableOutcome &outcome =
 dynamoClient.UpdateTable(
```

```
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated the table." << std::endl;
    } else {
        const Aws::DynamoDB::DynamoDBError &error = outcome.GetError();
        if (error.GetErrorType() == Aws::DynamoDB::DynamoDBErrors::VALIDATION &&
            error.GetMessage().find("The provisioned throughput for the table
 will not change") != std::string::npos) {
            std::cout << "The provisioned throughput for the table will not
 change." << std::endl;
        } else {
            std::cerr << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    return waitTableActive(tableName, dynamoClient);
}
```

Código que espera a que se active la tabla.

```
//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
```

```
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obtener información sobre la API, consulte UpdateTable en la Referencia de la API de AWS SDK para C++.

CLI

AWS CLI

Ejemplo 1: modificar el modo de facturación de una tabla

El siguiente ejemplo de `update-table` aumenta la capacidad de lectura y escritura aprovisionada en la tabla `MusicCollection`.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --billing-mode PROVISIONED \
    --provisioned-throughput ReadCapacityUnits=15,WriteCapacityUnits=10
```

Salida:

```
{
```

```
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T13:18:18.921000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        }
```

```
    }
}
```

Para obtener más información, consulte Actualización de una tabla en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 5: crear un índice secundario global

En el siguiente ejemplo se añade un índice secundario global a la tabla `MusicCollection`.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=AlbumTitle,AttributeType=S \
    --global-secondary-index-updates file://gsi-updates.json
```

Contenido de `gsi-updates.json`:

```
[
    {
        "Create": {
            "IndexName": "AlbumTitle-index",
            "KeySchema": [
                {
                    "AttributeName": "AlbumTitle",
                    "KeyType": "HASH"
                }
            ],
            "ProvisionedThroughput": {
                "ReadCapacityUnits": 10,
                "WriteCapacityUnits": 10
            },
            "Projection": {
                "ProjectionType": "ALL"
            }
        }
    }
]
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
```

```
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "GlobalSecondaryIndexes": [
            {
```

```
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "CREATING",
                "Backfilling": false,
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ]
    }
}
```

Para obtener más información, consulte Actualización de una tabla en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 3: activar DynamoDB Streams en una tabla

El siguiente comando activa DynamoDB Streams en la tabla `MusicCollection`.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_IMAGE
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
```

```
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "UPDATING",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
```

```
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
                    "NonKeyAttributes": [
                        "Year",
                        "Genre"
                    ]
                },
                "IndexSizeBytes": 139,
                "ItemCount": 2,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ],
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
```

```
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ],
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_IMAGE"
        },
        "LatestStreamLabel": "2020-07-28T21:53:39.112",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/stream/2020-07-28T21:53:39.112"
    }
}
```

Para obtener más información, consulte Actualización de una tabla en la Guía para desarrolladores de Amazon DynamoDB.

Ejemplo 4: activar el cifrado en el lado del servidor

El siguiente ejemplo activa el cifrado del lado del servidor en la tabla `MusicCollection`.

```
aws dynamodb update-table \
    --table-name MusicCollection \
    --sse-specification Enabled=true,SSEType=KMS
```

Salida:

```
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "AlbumTitle",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
```

```
        "TableName": "MusicCollection",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": "2020-05-26T15:59:49.473000-07:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "2020-07-28T12:59:17.537000-07:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 15,
            "WriteCapacityUnits": 10
        },
        "TableSizeBytes": 182,
        "ItemCount": 2,
        "TableArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection",
        "TableId": "abcd0123-01ab-23cd-0123-abcdef123456",
        "BillingModeSummary": {
            "BillingMode": "PROVISIONED",
            "LastUpdateToPayPerRequestDateTime":
 "2020-07-28T13:14:48.366000-07:00"
        },
        "LocalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitleIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    },
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "RANGE"
                    }
                ],
                "Projection": {
                    "ProjectionType": "INCLUDE",
```

```
                    "NonKeyAttributes": [
                        "Year",
                        "Genre"
                    ]
                },
                "IndexSizeBytes": 139,
                "ItemCount": 2,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitleIndex"
            }
        ],
        "GlobalSecondaryIndexes": [
            {
                "IndexName": "AlbumTitle-index",
                "KeySchema": [
                    {
                        "AttributeName": "AlbumTitle",
                        "KeyType": "HASH"
                    }
                ],
                "Projection": {
                    "ProjectionType": "ALL"
                },
                "IndexStatus": "ACTIVE",
                "ProvisionedThroughput": {
                    "NumberOfDecreasesToday": 0,
                    "ReadCapacityUnits": 10,
                    "WriteCapacityUnits": 10
                },
                "IndexSizeBytes": 0,
                "ItemCount": 0,
                "IndexArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/index/AlbumTitle-index"
            }
        ],
        "StreamSpecification": {
            "StreamEnabled": true,
            "StreamViewType": "NEW_IMAGE"
        },
        "LatestStreamLabel": "2020-07-28T21:53:39.112",
        "LatestStreamArn": "arn:aws:dynamodb:us-west-2:123456789012:table/
MusicCollection/stream/2020-07-28T21:53:39.112",
        "SSEDescription": {
            "Status": "UPDATING"
```

```
        }
    }
}
```

Para obtener más información, consulte Actualización de una tabla en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte UpdateTable en la AWS CLI Command Reference.

PowerShell

Herramientas para PowerShell V4

Ejemplo 1: actualiza el rendimiento aprovisionado de la tabla en cuestión.

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- Para obtener detalles de la API, consulte UpdateTable en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V4).

Herramientas para PowerShell V5

Ejemplo 1: actualiza el rendimiento aprovisionado de la tabla en cuestión.

```
Update-DDBTable -TableName "myTable" -ReadCapacity 10 -WriteCapacity 5
```

- Para obtener detalles de la API, consulte UpdateTable en la Referencia de cmdlets de Herramientas de AWS para PowerShell (V5).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de **`UpdateTimeToLive`** con un SDK de AWS o la CLI

Los siguientes ejemplos de código muestran cómo utilizar `UpdateTimeToLive`.

Los ejemplos de acciones son extractos de código de programas más grandes y deben ejecutarse en contexto. Puede ver esta acción en contexto en el siguiente ejemplo de código:

- Trabajo con Streams y tiempo de vida

CLI

AWS CLI

Actualización de la configuración de tiempo de vida en una tabla

El siguiente ejemplo de `update-time-to-live` activa Tiempo de vida en la tabla especificada.

```
aws dynamodb update-time-to-live \
    --table-name MusicCollection \
    --time-to-live-specification Enabled=true,AttributeName=ttl
```

Salida:

```
{
    "TimeToLiveSpecification": {
        "Enabled": true,
        "AttributeName": "ttl"
    }
}
```

Para obtener más información, consulte Tiempo de vida en la Guía para desarrolladores de Amazon DynamoDB.

- Para obtener información sobre la API, consulte UpdateTimeToLive en la Referencia de comandos de la AWS CLI.

Java

SDK para Java 2.x

Habilite TTL en una tabla de DynamoDB existente con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.TimeToLiveSpecification;
```

```
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveResponse;

import java.util.logging.Level;
import java.util.logging.Logger;

    public UpdateTimeToLiveResponse enableTTL(final String tableName, final
 String attributeName, final Region region) {
        final TimeToLiveSpecification ttlSpec = TimeToLiveSpecification.builder()
            .attributeName(attributeName)
            .enabled(true)
            .build();

        final UpdateTimeToLiveRequest request = UpdateTimeToLiveRequest.builder()
            .tableName(tableName)
            .timeToLiveSpecification(ttlSpec)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            return ddb.updateTimeToLive(request);
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

Deshabilite TTL en una tabla de DynamoDB existente con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.TimeToLiveSpecification;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateTimeToLiveResponse;

import java.util.logging.Level;
```

```
import java.util.logging.Logger;

    public UpdateTimeToLiveResponse disableTTL(
        final String tableName, final String attributeName, final Region region)
 {
        final TimeToLiveSpecification ttlSpec = TimeToLiveSpecification.builder()
            .attributeName(attributeName)
            .enabled(false)
            .build();

        final UpdateTimeToLiveRequest request = UpdateTimeToLiveRequest.builder()
            .tableName(tableName)
            .timeToLiveSpecification(ttlSpec)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            return ddb.updateTimeToLive(request);
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

- Para obtener información sobre la API, consulte UpdateTimeToLive en la Referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Habilite TTL en una tabla de DynamoDB existente.

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";

export const enableTTL = async (tableName, ttlAttribute, region = 'us-east-1') =>
 {
```

```
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: true,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL enabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to enable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error enabling TTL: ${e}`);
        throw e;
    }
};

// Example usage (commented out for testing)
// enableTTL('ExampleTable', 'exampleTtlAttribute');
```

Deshabilite TTL en una tabla de DynamoDB existente.

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-
dynamodb";

export const disableTTL = async (tableName, ttlAttribute, region = 'us-east-1')
 => {

    const client = new DynamoDBClient({
        region: region,
```

```
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const params = {
        TableName: tableName,
        TimeToLiveSpecification: {
            Enabled: false,
            AttributeName: ttlAttribute
        }
    };

    try {
        const response = await client.send(new UpdateTimeToLiveCommand(params));
        if (response.$metadata.httpStatusCode === 200) {
            console.log(`TTL disabled successfully for table ${tableName}, using
 attribute name ${ttlAttribute}.`);
        } else {
            console.log(`Failed to disable TTL for table ${tableName}, response
 object: ${response}`);
        }
        return response;
    } catch (e) {
        console.error(`Error disabling TTL: ${e}`);
        throw e;
    }
};

// Example usage (commented out for testing)
// disableTTL('ExampleTable', 'exampleTtlAttribute');
```

- Para obtener información sobre la API, consulte UpdateTimeToLive en la Referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Habilite TTL en una tabla de DynamoDB existente.

```
import boto3


def enable_ttl(table_name, ttl_attribute_name):
```

```
    """
    Enables TTL on DynamoDB table for a given attribute name
        on success, returns a status code of 200
        on error, throws an exception

    :param table_name: Name of the DynamoDB table
    :param ttl_attribute_name: The name of the TTL attribute being provided to
 the table.
    """
    try:
        dynamodb = boto3.client("dynamodb")

        # Enable TTL on an existing DynamoDB table
        response = dynamodb.update_time_to_live(
            TableName=table_name,
            TimeToLiveSpecification={"Enabled": True, "AttributeName":
 ttl_attribute_name},
        )

        # In the returned response, check for a successful status code.
        if response["ResponseMetadata"]["HTTPStatusCode"] == 200:
            print("TTL has been enabled successfully.")
        else:
            print(
                f"Failed to enable TTL, status code {response['ResponseMetadata']
['HTTPStatusCode']}"
            )
        return response
    except Exception as ex:
        print("Couldn't enable TTL in table %s. Here's why: %s" % (table_name,
 ex))
        raise


# your values
enable_ttl("your-table-name", "expireAt")
```

Deshabilite TTL en una tabla de DynamoDB existente.

```
import boto3
```

```
def disable_ttl(table_name, ttl_attribute_name):
    """
    Disables TTL on DynamoDB table for a given attribute name
        on success, returns a status code of 200
        on error, throws an exception

    :param table_name: Name of the DynamoDB table being modified
    :param ttl_attribute_name: The name of the TTL attribute being provided to
 the table.
    """
    try:
        dynamodb = boto3.client("dynamodb")

        # Enable TTL on an existing DynamoDB table
        response = dynamodb.update_time_to_live(
            TableName=table_name,
            TimeToLiveSpecification={"Enabled": False, "AttributeName":
 ttl_attribute_name},
        )

        # In the returned response, check for a successful status code.
        if response["ResponseMetadata"]["HTTPStatusCode"] == 200:
            print("TTL has been disabled successfully.")
        else:
            print(
                f"Failed to disable TTL, status code
 {response['ResponseMetadata']['HTTPStatusCode']}"
            )
    except Exception as ex:
        print("Couldn't disable TTL in table %s. Here's why: %s" % (table_name,
 ex))
        raise


# your values
disable_ttl("your-table-name", "expireAt")
```

- Para obtener información sobre la API, consulte UpdateTimeToLive en la Referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Escenarios para DynamoDB con los SDK de AWS

En los siguientes ejemplos de código se muestra cómo implementar escenarios comunes en DynamoDB con los SDK de AWS. Estos escenarios muestran cómo llevar a cabo tareas específicas con llamadas a varias funciones dentro de DynamoDB o en combinación con otros Servicios de AWS. En cada escenario se incluye un enlace al código fuente completo, con instrucciones de configuración y ejecución del código.

Los escenarios requieren un nivel intermedio de experiencia para ayudarlo a entender las acciones de servicio en su contexto.

Ejemplos

- Aceleración de lecturas de DynamoDB con DAX con un SDK de AWS
- Trabajo con escenarios avanzados de índice secundario global de DynamoDB mediante AWS Command Line Interface v2
- Creación de una aplicación para enviar datos a una tabla de DynamoDB
- Comparación de varios valores con un solo atributo en DynamoDB con un AWS SDK
- Actualización condicional de un elemento de DynamoDB con un TTL mediante un AWS SDK
- Conexión a una instancia local de DynamoDB mediante un SDK de AWS
- Recuento de operadores de expresión en DynamoDB con un AWS SDK
- Creación de una API REST de API Gateway para realizar un seguimiento de datos de COVID-19
- Creación de una aplicación de mensajería con Step Functions
- Creación de una aplicación de administración de activos fotográficos que permita a los usuarios administrar las fotos mediante etiquetas
- Creación de una tabla de DynamoDB con un índice secundario global mediante el AWS SDK
- Creación de una tabla de DynamoDB con configuración de rendimiento en caliente utilizando un AWS SDK
- Creación de una aplicación web para hacer un seguimiento de los datos de DynamoDB
- Creación una aplicación de chat de websocket con API Gateway
- Creación de un elemento de DynamoDB con un TTL mediante un AWS SDK

- Creación y administración de tablas globales de DynamoDB con coherencia alta de varias regiones mediante un AWS SDK
- Creación y administración de tablas globales de DynamoDB que muestren MREC con un AWS SDK
- Eliminación de datos de DynamoDB mediante instrucciones DELETE de PartiQL con un AWS SDK
- Detección de EPI en imágenes con Amazon Rekognition mediante un AWS SDK
- Inserción de datos de DynamoDB mediante instrucciones INSERT de PartiQL con un AWS SDK
- Invocación de una función Lambda desde un navegador
- Administración de índices secundarios globales de DynamoDB mediante AWS Command Line Interface v2
- Administración de las políticas basadas en recursos de DynamoDB mediante AWS Command Line Interface v2
- Monitoreo del rendimiento de Amazon DynamoDB mediante un SDK de AWS
- Realización de operaciones de consulta avanzadas de DynamoDB mediante un SDK de AWS
- Realización de operaciones de lista en DynamoDB con un AWS SDK
- Realización de operaciones de asignación en DynamoDB con un AWS SDK
- Realización de operaciones de conjunto en DynamoDB con un AWS SDK
- Consultar una tabla de DynamoDB mediante lotes de instrucciones PartiQL y un SDK de AWS
- Consulta de una tabla de DynamoDB con PartiQL y un SDK de AWS
- Consulta de una tabla de DynamoDB mediante un índice secundario global con un SDK de AWS
- Consulta de una tabla de DynamoDB mediante una condición begins_with con un SDK de AWS
- Consulta de una tabla de DynamoDB mediante un intervalo de fechas en la clave de clasificación con un SDK de AWS
- Consulta de una tabla de DynamoDB con una expresión de filtro compleja con un SDK de AWS
- Consulta de una tabla de DynamoDB con una expresión de filtro compleja con un SDK de AWS
- Consulta de una tabla de DynamoDB con una expresión de filtro y límite con un SDK de AWS
- Consulta de una tabla de DynamoDB con atributos anidados mediante un SDK de AWS
- Consulta de una tabla de DynamoDB con paginación mediante un SDK de AWS
- Consulta de una tabla de DynamoDB con lecturas altamente coherentes mediante un SDK de AWS
- Consulta de datos de DynamoDB mediante instrucciones SELECT de PartiQL con un AWS SDK

- Consulta de elementos de TTL en una tabla de DynamoDB con un AWS SDK
- Consulta de tablas de DynamoDB mediante patrones de fecha y hora con un SDK de AWS
- Guarde EXIF y otra información de la imagen con un SDK de AWS
- Configuración del control de acceso basado en atributos para DynamoDB mediante AWS Command Line Interface v2
- Descripción del orden de las expresiones de actualización en DynamoDB con un AWS SDK
- Actualización de una configuración de tabla de DynamoDB con rendimiento en caliente mediante un AWS SDK
- Actualización de un elemento de DynamoDB con un TTL mediante un AWS SDK
- Actualización de datos de DynamoDB mediante instrucciones UPDATE de PartiQL con un AWS SDK
- Uso de API Gateway para invocar una función de Lambda
- Uso de Step Functions para invocar funciones de Lambda
- Utilizar un modelo de documento para DynamoDB mediante un AWS SDK
- Utilizar un modelo de persistencia de objetos de alto nivel para DynamoDB mediante un AWS SDK
- Uso de operaciones de contador atómico en DynamoDB con un AWS SDK
- Uso de operaciones condicionales en DynamoDB con un AWS SDK
- Uso de nombres de atributo de expresión en DynamoDB con un AWS SDK
- Uso de eventos programados para invocar una función de Lambda
- Trabajo con índices secundarios locales de DynamoDB mediante AWS Command Line Interface v2
- Trabajo con DynamoDB Streams y tiempo de vida mediante AWS Command Line Interface v2
- Trabajo con tablas globales de DynamoDB y replicación de varias regiones con coherencia final (MREC) mediante AWS Command Line Interface v2
- Trabajo con el etiquetado de recursos de DynamoDB mediante AWS Command Line Interface v2
- Trabajo con el cifrado de tablas de DynamoDB mediante AWS Command Line Interface v2

Aceleración de lecturas de DynamoDB con DAX con un SDK de AWS

En el siguiente ejemplo de código, se muestra cómo:

- Cree y escriba datos en una tabla con los clientes de DAX y SDK.

- Obtenga, consulte y explore la tabla con ambos clientes y compare su rendimiento.

Para obtener más información, consulte Desarrollo con el cliente de DynamoDB Accelerator.

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear una tabla con el cliente de DAX o Boto3.

```
import boto3


def create_dax_table(dyn_resource=None):
    """
    Creates a DynamoDB table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The newly created table.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table_name = "TryDaxTable"
    params = {
        "TableName": table_name,
        "KeySchema": [
            {"AttributeName": "partition_key", "KeyType": "HASH"},
            {"AttributeName": "sort_key", "KeyType": "RANGE"},
        ],
        "AttributeDefinitions": [
            {"AttributeName": "partition_key", "AttributeType": "N"},
            {"AttributeName": "sort_key", "AttributeType": "N"},
        ],
        "BillingMode": "PAY_PER_REQUEST",
    }
```

```
    table = dyn_resource.create_table(**params)
    print(f"Creating {table_name}...")
    table.wait_until_exists()
    return table


if __name__ == "__main__":
    dax_table = create_dax_table()
    print(f"Created table.")
```

Escribir datos de prueba en la tabla.

```
import boto3


def write_data_to_dax_table(key_count, item_size, dyn_resource=None):
    """
    Writes test data to the demonstration table.

    :param key_count: The number of partition and sort keys to use to populate
 the
                      table. The total number of items is key_count * key_count.
    :param item_size: The size of non-key data for each test item.
    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    some_data = "X" * item_size

    for partition_key in range(1, key_count + 1):
        for sort_key in range(1, key_count + 1):
            table.put_item(
                Item={
                    "partition_key": partition_key,
                    "sort_key": sort_key,
                    "some_data": some_data,
                }
            )
            print(f"Put item ({partition_key}, {sort_key}) succeeded.")
```

```
if __name__ == "__main__":
    write_key_count = 10
    write_item_size = 1000
    print(
        f"Writing {write_key_count*write_key_count} items to the table. "
        f"Each item is {write_item_size} characters."
    )
    write_data_to_dax_table(write_key_count, write_item_size)
```

Obtener elementos para una serie de iteraciones tanto para el cliente de DAX como para el cliente de Boto3 e informar del tiempo empleado en cada uno.

```
import argparse
import sys
import time
import amazondax
import boto3


def get_item_test(key_count, iterations, dyn_resource=None):
    """
    Gets items from the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param key_count: The number of items to get from the table in each
 iteration.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        for partition_key in range(1, key_count + 1):
            for sort_key in range(1, key_count + 1):
                table.get_item(
                    Key={"partition_key": partition_key, "sort_key": sort_key}
```

```
                )
                print(".", end="")
                sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
    )
    args = parser.parse_args()

    test_key_count = 10
    test_iterations = 50
    if args.endpoint_url:
        print(
            f"Getting each item from the table {test_iterations} times, "
            f"using the DAX client."
        )
        # Use a with statement so the DAX client closes the cluster after
 completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
 as dax:
            test_start, test_end = get_item_test(
                test_key_count, test_iterations, dyn_resource=dax
            )
    else:
        print(
            f"Getting each item from the table {test_iterations} times, "
            f"using the Boto3 client."
        )
        test_start, test_end = get_item_test(test_key_count, test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/ test_iterations}."
    )
```

Consultar la tabla durante una serie de iteraciones tanto para el cliente de DAX como para el cliente de Boto3 e informar del tiempo empleado en cada uno.

```
import argparse
import time
import sys
import amazondax
import boto3
from boto3.dynamodb.conditions import Key


def query_test(partition_key, sort_keys, iterations, dyn_resource=None):
    """
    Queries the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param partition_key: The partition key value to use in the query. The query
                          returns items that have partition keys equal to this
 value.
    :param sort_keys: The range of sort key values for the query. The query
 returns
                      items that have sort key values between these two values.
    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    key_condition_expression = Key("partition_key").eq(partition_key) & Key(
        "sort_key"
    ).between(*sort_keys)

    start = time.perf_counter()
    for _ in range(iterations):
        table.query(KeyConditionExpression=key_condition_expression)
        print(".", end="")
        sys.stdout.flush()
    print()
```

```
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
    )
    args = parser.parse_args()

    test_partition_key = 5
    test_sort_keys = (2, 9)
    test_iterations = 100
    if args.endpoint_url:
        print(f"Querying the table {test_iterations} times, using the DAX
 client.")
        # Use a with statement so the DAX client closes the cluster after
 completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
 as dax:
            test_start, test_end = query_test(
                test_partition_key, test_sort_keys, test_iterations,
 dyn_resource=dax
            )
    else:
        print(f"Querying the table {test_iterations} times, using the Boto3
 client.")
        test_start, test_end = query_test(
            test_partition_key, test_sort_keys, test_iterations
        )

    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

Examinar la tabla durante una serie de iteraciones tanto para el cliente de DAX como para el cliente de Boto3 e informar del tiempo empleado en cada uno.

```
import argparse
import time
import sys
import amazondax
import boto3


def scan_test(iterations, dyn_resource=None):
    """
    Scans the table a specified number of times. The time before the
    first iteration and the time after the last iteration are both captured
    and reported.

    :param iterations: The number of iterations to run.
    :param dyn_resource: Either a Boto3 or DAX resource.
    :return: The start and end times of the test.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    start = time.perf_counter()
    for _ in range(iterations):
        table.scan()
        print(".", end="")
        sys.stdout.flush()
    print()
    end = time.perf_counter()
    return start, end


if __name__ == "__main__":
    # pylint: disable=not-context-manager
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "endpoint_url",
        nargs="?",
        help="When specified, the DAX cluster endpoint. Otherwise, DAX is not
 used.",
    )
```

```
    args = parser.parse_args()

    test_iterations = 100
    if args.endpoint_url:
        print(f"Scanning the table {test_iterations} times, using the DAX
 client.")
        # Use a with statement so the DAX client closes the cluster after
 completion.
        with amazondax.AmazonDaxClient.resource(endpoint_url=args.endpoint_url)
 as dax:
            test_start, test_end = scan_test(test_iterations, dyn_resource=dax)
    else:
        print(f"Scanning the table {test_iterations} times, using the Boto3
 client.")
        test_start, test_end = scan_test(test_iterations)
    print(
        f"Total time: {test_end - test_start:.4f} sec. Average time: "
        f"{(test_end - test_start)/test_iterations}."
    )
```

Elimine la tabla .

```
import boto3


def delete_dax_table(dyn_resource=None):
    """
    Deletes the demonstration table.

    :param dyn_resource: Either a Boto3 or DAX resource.
    """
    if dyn_resource is None:
        dyn_resource = boto3.resource("dynamodb")

    table = dyn_resource.Table("TryDaxTable")
    table.delete()

    print(f"Deleting {table.name}...")
    table.wait_until_not_exists()


if __name__ == "__main__":
```

```
    delete_dax_table()
    print("Table deleted!")
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Python (Boto3).
  - CreateTable
  - DeleteTable
  - GetItem
  - PutItem
  - Query
  - Examen

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Trabajo con escenarios avanzados de índice secundario global de DynamoDB mediante AWS Command Line Interface v2

En el siguiente ejemplo de código se muestra cómo trabajar con configuraciones avanzadas de índice secundario global.

- Cree una tabla con varios GSI.
- Cree una tabla con capacidad bajo demanda y GSI.
- Coloque elementos en una tabla con múltiples GSI.
- Consultar múltiples GSI con diferentes condiciones.

Bash

AWS CLI con Bash script

Cree una tabla con varios GSI.

```
# Create a table with multiple GSIs
aws dynamodb create-table \
```

```
    --table-name MusicLibrary \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
        AttributeName=AlbumTitle,AttributeType=S \
        AttributeName=Genre,AttributeType=S \
        AttributeName=Year,AttributeType=N \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"AlbumIndex\",
                \"KeySchema\": [{\"AttributeName\":\"AlbumTitle\",\"KeyType\":
\"HASH\"}],
                \"Projection\": {\"ProjectionType\":\"ALL\"}
            },
            {
                \"IndexName\": \"GenreYearIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"Genre\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"Year\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {\"ProjectionType\":\"INCLUDE\",
\"NonKeyAttributes\":[\"Artist\",\"SongTitle\"]}
            }
        ]"
```

Cree una tabla con capacidad bajo demanda y GSI.

```
# Create a table with on-demand capacity and GSI
aws dynamodb create-table \
    --table-name MusicOnDemand \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
        AttributeName=Genre,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
```

```
    --billing-mode PAY_PER_REQUEST \
    --global-secondary-indexes \
        "[
            {
                \"IndexName\": \"GenreIndex\",
                \"KeySchema\": [{\"AttributeName\":\"Genre\",\"KeyType\":\"HASH
\"}],
                \"Projection\": {\"ProjectionType\":\"ALL\"}
            }
        ]"
```

Coloque elementos en una tabla con múltiples GSI.

```
# Add items to MusicLibrary table
aws dynamodb put-item \
    --table-name MusicLibrary \
    --item '{
        "Artist": {"S": "The Beatles"},
        "SongTitle": {"S": "Hey Jude"},
        "AlbumTitle": {"S": "Past Masters"},
        "Genre": {"S": "Rock"},
        "Year": {"N": "1968"}
    }'

aws dynamodb put-item \
    --table-name MusicLibrary \
    --item '{
        "Artist": {"S": "Miles Davis"},
        "SongTitle": {"S": "So What"},
        "AlbumTitle": {"S": "Kind of Blue"},
        "Genre": {"S": "Jazz"},
        "Year": {"N": "1959"}
    }'
```

Consulte elementos de una tabla con múltiples GSI.

```
# Query the AlbumIndex GSI
echo "Querying AlbumIndex GSI:"
aws dynamodb query \
    --table-name MusicLibrary \
    --index-name AlbumIndex \
```

```
    --key-condition-expression "AlbumTitle = :album" \
    --expression-attribute-values '{":album":{"S":"Kind of Blue"}}'

# Query the GenreYearIndex GSI with a range condition
echo "Querying GenreYearIndex GSI with range condition:"
aws dynamodb query \
    --table-name MusicLibrary \
    --index-name GenreYearIndex \
    --key-condition-expression "Genre = :genre AND #yr > :year" \
    --expression-attribute-names '{"#yr": "Year"}' \
    --expression-attribute-values '{":genre":{"S":"Rock"},":year":{"N":"1965"}}'
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de comandos de AWS CLI.
  - CreateTable
  - PutItem
  - Query

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Creación de una aplicación para enviar datos a una tabla de DynamoDB

Los siguientes ejemplos de código indican cómo crear una aplicación que envíe datos a una tabla de Amazon DynamoDB y que le notifique cuando un usuario actualice la tabla.

Java

SDK para Java 2.x

Indica cómo crear una aplicación web dinámica que envíe datos mediante la API Java de Amazon DynamoDB y un mensaje de texto mediante la API Java de Amazon Simple Notification Service.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- DynamoDB
- Amazon SNS

JavaScript

SDK para JavaScript (v3)

Este ejemplo indica cómo crear una aplicación que permita a los usuarios enviar datos a una tabla de Amazon DynamoDB y un mensaje de texto al administrador mediante Amazon Simple Notification Service (Amazon SNS).

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Este ejemplo también está disponible en la Guía para desarrolladores de AWS SDK para JavaScript v3.

Servicios utilizados en este ejemplo

- DynamoDB
- Amazon SNS

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Comparación de varios valores con un solo atributo en DynamoDB con un AWS SDK

En los siguientes ejemplos de código se muestra cómo comparar varios valores con un solo atributo en DynamoDB.

- Utilice el operador IN para comparar varios valores con un solo atributo.
- Compare el operador IN con varias condiciones OR.
- Conozca las ventajas de rendimiento y complejidad de las expresiones que ofrece el uso de IN.

Java

SDK para Java 2.x

Compare varios valores con un solo atributo en DynamoDB con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;

    /**
     * Queries a table using the IN operator to compare multiple values with a
 single attribute.
     *
     * <p>This method demonstrates how to use the IN operator in a filter
 expression
     * to match an attribute against multiple values.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key to query
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return The query response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static QueryResponse compareMultipleValues(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
        AttributeValue partitionKeyValue,
        String attributeName,
        List<AttributeValue> valuesList) {
```

```
        // Create expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pkName", partitionKeyName);
        expressionAttributeNames.put("#attrName", attributeName);

        // Create expression attribute values
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":pkValue", partitionKeyValue);

        // Add values for IN operator
        for (int i = 0; i < valuesList.size(); i++) {
            expressionAttributeValues.put(":val" + i, valuesList.get(i));
        }

        // Build the IN clause
        StringBuilder inClause = new StringBuilder();
        for (int i = 0; i < valuesList.size(); i++) {
            if (i > 0) {
                inClause.append(", ");
            }
            inClause.append(":val").append(i);
        }

        // Define the query parameters
        QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression("#pkName = :pkValue")
            .filterExpression("#attrName IN (" + inClause.toString() + ")")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the query operation
        return dynamoDbClient.query(request);
    }

    /**
     * Queries a table using multiple OR conditions to compare multiple values
with a single attribute.
     *
     * <p>This method demonstrates the alternative approach to using the IN
operator,
     * by using multiple OR conditions.
```

```
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key to query
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return The query response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static QueryResponse compareWithOrConditions(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
        AttributeValue partitionKeyValue,
        String attributeName,
        List<AttributeValue> valuesList) {

        // Create expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pkName", partitionKeyName);
        expressionAttributeNames.put("#attrName", attributeName);

        // Create expression attribute values
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":pkValue", partitionKeyValue);

        // Add values for OR conditions
        for (int i = 0; i < valuesList.size(); i++) {
            expressionAttributeValues.put(":val" + i, valuesList.get(i));
        }

        // Build the OR conditions
        StringBuilder orConditions = new StringBuilder();
        for (int i = 0; i < valuesList.size(); i++) {
            if (i > 0) {
                orConditions.append(" OR ");
            }
            orConditions.append("#attrName = :val").append(i);
        }

        // Define the query parameters
        QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
```

```
            .keyConditionExpression("#pkName = :pkValue")
            .filterExpression(orConditions.toString())
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the query operation
        return dynamoDbClient.query(request);
    }

    /**
     * Compares the performance of using the IN operator versus multiple OR
conditions.
     *
     * <p>This method demonstrates the performance difference between using the
IN operator
     * and using multiple OR conditions.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key to query
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return Map containing the performance comparison results
     */
    public static Map<String, Object> comparePerformance(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
        AttributeValue partitionKeyValue,
        String attributeName,
        List<AttributeValue> valuesList) {

        Map<String, Object> results = new HashMap<>();

        try {
            // Measure performance of IN operator
            long inStartTime = System.nanoTime();
            QueryResponse inResponse = compareMultipleValues(
                dynamoDbClient, tableName, partitionKeyName, partitionKeyValue,
attributeName, valuesList);
            long inEndTime = System.nanoTime();
            long inDuration = inEndTime - inStartTime;
```

```
            // Measure performance of OR conditions
            long orStartTime = System.nanoTime();
            QueryResponse orResponse = compareWithOrConditions(
                dynamoDbClient, tableName, partitionKeyName, partitionKeyValue,
attributeName, valuesList);
            long orEndTime = System.nanoTime();
            long orDuration = orEndTime - orStartTime;

            // Record results
            results.put("inOperatorDuration", inDuration);
            results.put("orConditionsDuration", orDuration);
            results.put("inOperatorItems", inResponse.count());
            results.put("orConditionsItems", orResponse.count());
            results.put("inOperatorExpression", "IN operator with " +
valuesList.size() + " values");
            results.put("orConditionsExpression", valuesList.size() + " OR
conditions");
            results.put("success", true);

        } catch (DynamoDbException e) {
            results.put("success", false);
            results.put("error", e.getMessage());
        }

        return results;
    }

    /**
     * Scans a table using the IN operator with a large number of values.
     *
     * <p>This method demonstrates how to use the IN operator with a large number
of values,
     * which can help stay within the 300 operator limit.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param attributeName The name of the attribute to compare
     * @param valuesList List of values to compare against
     * @return The scan response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static ScanResponse scanWithLargeInClause(
```

```
        DynamoDbClient dynamoDbClient, String tableName, String attributeName,
List<AttributeValue> valuesList) {

        // Create expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#attrName", attributeName);

        // Create expression attribute values
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();

        // Add values for IN operator
        for (int i = 0; i < valuesList.size(); i++) {
            expressionAttributeValues.put(":val" + i, valuesList.get(i));
        }

        // Build the IN clause
        StringBuilder inClause = new StringBuilder();
        for (int i = 0; i < valuesList.size(); i++) {
            if (i > 0) {
                inClause.append(", ");
            }
            inClause.append(":val").append(i);
        }

        // Define the scan parameters
        ScanRequest request = ScanRequest.builder()
            .tableName(tableName)
            .filterExpression("#attrName IN (" + inClause.toString() + ")")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the scan operation
        return dynamoDbClient.scan(request);
    }

    /**
     * Generates a list of sample values for testing.
     *
     * <p>Helper method to generate a list of sample values for testing.
     *
     * @param valueType The type of values to generate (string, number, or
boolean)
     * @param count The number of values to generate
```

```
     * @return List of generated attribute values
     */
    public static List<AttributeValue> generateSampleValues(String valueType, int
count) {
        List<AttributeValue> values = new ArrayList<>();

        for (int i = 0; i < count; i++) {
            AttributeValue value;

            switch (valueType.toLowerCase(Locale.ROOT)) {
                case "string":
                    value = AttributeValue.builder().s("Value" + i).build();
                    break;
                case "number":
                    value =
AttributeValue.builder().n(String.valueOf(i)).build();
                    break;
                case "boolean":
                    value = AttributeValue.builder().bool(i % 2 == 0).build();
                    break;
                default:
                    throw new IllegalArgumentException("Unsupported value type: "
+ valueType);
            }

            values.add(value);
        }

        return values;
    }
```

Ejemplo de uso de la comparación de varios valores con AWS SDK for Java 2.x.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        System.out.println("Demonstrating how to compare multiple values with a
single attribute in DynamoDB");

        try {
            // Example 1: Using the IN operator
            System.out.println("\nExample 1: Using the IN operator");
            List<AttributeValue> categories = List.of(
```

```
                AttributeValue.builder().s("Electronics").build(),
                AttributeValue.builder().s("Computers").build(),
                AttributeValue.builder().s("Accessories").build());

            QueryResponse inResponse = compareMultipleValues(
                dynamoDbClient,
                tableName,
                "Department",
                AttributeValue.builder().s("Retail").build(),
                "Category",
                categories);

            System.out.println("Found " + inResponse.count() + " items using IN
operator");
            System.out.println("Items: " + inResponse.items());

            // Example 2: Using multiple OR conditions
            System.out.println("\nExample 2: Using multiple OR conditions");
            QueryResponse orResponse = compareWithOrConditions(
                dynamoDbClient,
                tableName,
                "Department",
                AttributeValue.builder().s("Retail").build(),
                "Category",
                categories);

            System.out.println("Found " + orResponse.count() + " items using OR
conditions");
            System.out.println("Items: " + orResponse.items());

            // Example 3: Performance comparison
            System.out.println("\nExample 3: Performance comparison");
            Map<String, Object> perfComparison = comparePerformance(
                dynamoDbClient,
                tableName,
                "Department",
                AttributeValue.builder().s("Retail").build(),
                "Category",
                categories);

            if ((boolean) perfComparison.get("success")) {
                System.out.println("IN operator duration: " +
perfComparison.get("inOperatorDuration") + " ns");
```

```
                System.out.println("OR conditions duration: " +
perfComparison.get("orConditionsDuration") + " ns");
                System.out.println("IN operator found " +
perfComparison.get("inOperatorItems") + " items");
                System.out.println("OR conditions found " +
perfComparison.get("orConditionsItems") + " items");
                System.out.println("Expression complexity comparison:");
                System.out.println("  IN operator: " +
perfComparison.get("inOperatorExpression"));
                System.out.println("  OR conditions: " +
perfComparison.get("orConditionsExpression"));
            } else {
                System.out.println("Performance comparison failed: " +
perfComparison.get("error"));
            }

            // Example 4: Using IN with a large number of values
            System.out.println("\nExample 4: Using IN with a large number of
values");
            List<AttributeValue> productIds = generateSampleValues("string", 20);

            ScanResponse largeInResponse = scanWithLargeInClause(dynamoDbClient,
tableName, "ProductId", productIds);

            System.out.println(
                "Found " + largeInResponse.count() + " items using IN with " +
productIds.size() + " values");

            // Explain the benefits of using IN
            System.out.println("\nKey points about using the IN operator in
DynamoDB:");
            System.out.println("1. The IN operator allows comparing a single
attribute against multiple values");
            System.out.println("2. IN is more concise than using multiple OR
conditions");
            System.out.println("3. IN counts as only 1 operator regardless of the
number of values");
            System.out.println("4. Multiple OR conditions count as 1 operator per
condition plus 1 per OR");
            System.out.println("5. Using IN helps stay within the 300 operator
limit for complex expressions");
            System.out.println("6. IN can be used in filter expressions and
condition expressions");
```

```
            System.out.println("7. The IN operator supports up to 100 comparison
 values");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for Java 2.x.
  - Query
  - Examen

JavaScript

SDK para JavaScript (v3)

Compare varios valores con un solo atributo con AWS SDK para JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  ScanCommand,
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Query or scan a DynamoDB table to find items where an attribute matches any
 value from a list.
 *
 * This function demonstrates the use of the IN operator to compare a single
 attribute
 * against multiple possible values, which is more efficient than using multiple
 OR conditions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against
 the values list
 * @param {Array} valuesList - List of values to compare the attribute against
```

```
 * @param {string} [partitionKeyName] - Optional name of the partition key
 attribute for query operations
 * @param {string} [partitionKeyValue] - Optional value of the partition key to
 query
 * @returns {Promise<Object>} - The response from DynamoDB containing the
 matching items
 */
async function compareMultipleValues(
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the filter expression using the IN operator
  const filterExpression = `${attributeName} IN (${valuesList.map((_, index) =>
 `:val${index}`).join(', ')})`;

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[`:val${index}`] = val;
    return acc;
  }, {});

  // If partition key is provided, perform a query operation
  if (partitionKeyName && partitionKeyValue) {
    const keyCondition = `${partitionKeyName} = :partitionKey`;
    expressionAttributeValues[':partitionKey'] = partitionKeyValue;

    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        KeyConditionExpression: keyCondition,
        FilterExpression: filterExpression,
```

```
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous
query
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new QueryCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  } else {
    // Otherwise, perform a scan operation
    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }
```

```
      const response = await docClient.send(new ScanCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  }
}

/**
 * Alternative implementation using multiple OR conditions instead of the IN
 operator.
 *
 * This function is provided for comparison to show why using the IN operator is
 preferable.
 * With many values, this approach becomes verbose and less efficient.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against
 the values list
 * @param {Array} valuesList - List of values to compare the attribute against
 * @param {string} [partitionKeyName] - Optional name of the partition key
 attribute for query operations
 * @param {string} [partitionKeyValue] - Optional value of the partition key to
 query
 * @returns {Promise<Object>} - The response from DynamoDB containing the
 matching items
 */
async function compareWithOrConditions(
  config,
  tableName,
  attributeName,
  valuesList,
```

```
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // If no values provided, return empty result
  if (!valuesList || valuesList.length === 0) {
    return {
      Items: [],
      Count: 0
    };
  }

  // Create the filter expression using multiple OR conditions
  const filterConditions = valuesList.map((_, index) => `${attributeName} = :val
${index}`);
  const filterExpression = filterConditions.join(' OR ');

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[`:val${index}`] = val;
    return acc;
  }, {});

  // If partition key is provided, perform a query operation
  if (partitionKeyName && partitionKeyValue) {
    const keyCondition = `${partitionKeyName} = :partitionKey`;
    expressionAttributeValues[':partitionKey'] = partitionKeyValue;

    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        KeyConditionExpression: keyCondition,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };
```

```
      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous
query
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new QueryCommand(params));

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  } else {
    // Otherwise, perform a scan operation
    // Initialize array to collect all items
    let allItems = [];
    let lastEvaluatedKey;

    // Use pagination to get all results
    do {
      const params = {
        TableName: tableName,
        FilterExpression: filterExpression,
        ExpressionAttributeValues: expressionAttributeValues
      };

      // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
      if (lastEvaluatedKey) {
        params.ExclusiveStartKey = lastEvaluatedKey;
      }

      const response = await docClient.send(new ScanCommand(params));

      // Add the items from this page to our collection
```

```
      if (response.Items && response.Items.length > 0) {
        allItems = [...allItems, ...response.Items];
      }

      // Get the key for the next page of results
      lastEvaluatedKey = response.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    // Return the complete result
    return {
      Items: allItems,
      Count: allItems.length
    };
  }
}
```

Ejemplo de uso de la comparación de varios valores con AWS SDK para JavaScript.

```
/**
 * Example of how to use the compareMultipleValues function.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const attributeName = "Category";
  const valuesList = ["Electronics", "Computers", "Accessories"];

  console.log(`Searching for products in any of these categories:
 ${valuesList.join(', ')}`);

  try {
    // Using the IN operator (recommended approach)
    console.log("\nApproach 1: Using the IN operator");
    const response = await compareMultipleValues(
      config,
      tableName,
      attributeName,
      valuesList
    );

    console.log(`Found ${response.Count} products in the specified categories`);
```

```
    // Using multiple OR conditions (alternative approach)
    console.log("\nApproach 2: Using multiple OR conditions");
    const response2 = await compareWithOrConditions(
      config,
      tableName,
      attributeName,
      valuesList
    );

    console.log(`Found ${response2.Count} products in the specified categories`);

    // Example with a query operation
    console.log("\nQuerying a specific manufacturer's products in multiple
 categories");
    const partitionKeyName = "Manufacturer";
    const partitionKeyValue = "Acme";

    const response3 = await compareMultipleValues(
      config,
      tableName,
      attributeName,
      valuesList,
      partitionKeyName,
      partitionKeyValue
    );

    console.log(`Found ${response3.Count} Acme products in the specified
 categories`);

    // Explain the benefits of using the IN operator
    console.log("\nBenefits of using the IN operator:");
    console.log("1. More concise expression compared to multiple OR conditions");
    console.log("2. Better readability and maintainability");
    console.log("3. Potentially better performance with large value lists");
    console.log("4. Simpler code that's less prone to errors");
    console.log("5. Easier to modify when adding or removing values");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para JavaScript.
  - Query
  - Examen

Python

SDK para Python (Boto3)

Compare varios valores con un solo atributo con AWS SDK for Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key
from typing import Any, Dict, List, Optional


def compare_multiple_values(
    table_name: str,
    attribute_name: str,
    values_list: List[Any],
    partition_key_name: Optional[str] = None,
    partition_key_value: Optional[str] = None,
) -> Dict[str, Any]:
    """
    Query or scan a DynamoDB table to find items where an attribute matches any
 value from a list.

    This function demonstrates the use of the IN operator to compare a single
 attribute
    against multiple possible values, which is more efficient than using multiple
 OR conditions.

    Args:
        table_name (str): The name of the DynamoDB table.
        attribute_name (str): The name of the attribute to compare against the
 values list.
        values_list (List[Any]): List of values to compare the attribute against.
        partition_key_name (Optional[str]): The name of the partition key
 attribute for query operations.
        partition_key_value (Optional[str]): The value of the partition key to
 query.
```

```
    Returns:
        Dict[str, Any]: The response from DynamoDB containing the matching items.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Create the filter expression using the is_in method
    filter_expression = Attr(attribute_name).is_in(values_list)

    # If partition key is provided, perform a query operation
    if partition_key_name and partition_key_value:
        key_condition = Key(partition_key_name).eq(partition_key_value)
        response = table.query(
            KeyConditionExpression=key_condition,
 FilterExpression=filter_expression
        )
    else:
        # Otherwise, perform a scan operation
        response = table.scan(FilterExpression=filter_expression)

    # Handle pagination if there are more results
    items = response.get("Items", [])
    while "LastEvaluatedKey" in response:
        if partition_key_name and partition_key_value:
            response = table.query(
                KeyConditionExpression=key_condition,
                FilterExpression=filter_expression,
                ExclusiveStartKey=response["LastEvaluatedKey"],
            )
        else:
            response = table.scan(
                FilterExpression=filter_expression,
 ExclusiveStartKey=response["LastEvaluatedKey"]
            )
        items.extend(response.get("Items", []))

    # Return the complete result
    return {"Items": items, "Count": len(items)}


def compare_with_or_conditions(
    table_name: str,
    attribute_name: str,
```

```
    values_list: List[Any],
    partition_key_name: Optional[str] = None,
    partition_key_value: Optional[str] = None,
) -> Dict[str, Any]:
    """
    Alternative implementation using multiple OR conditions instead of the IN
 operator.

    This function is provided for comparison to show why using the IN operator is
 preferable.
    With many values, this approach becomes verbose and less efficient.

    Args:
        table_name (str): The name of the DynamoDB table.
        attribute_name (str): The name of the attribute to compare against the
 values list.
        values_list (List[Any]): List of values to compare the attribute against.
        partition_key_name (Optional[str]): The name of the partition key
 attribute for query operations.
        partition_key_value (Optional[str]): The value of the partition key to
 query.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the matching items.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Create a filter expression with multiple OR conditions
    filter_expression = None
    for value in values_list:
        condition = Attr(attribute_name).eq(value)
        if filter_expression is None:
            filter_expression = condition
        else:
            filter_expression = filter_expression | condition

    # If partition key is provided, perform a query operation
    if partition_key_name and partition_key_value and filter_expression:
        key_condition = Key(partition_key_name).eq(partition_key_value)
        response = table.query(
            KeyConditionExpression=key_condition,
 FilterExpression=filter_expression
```

```
        )
    elif filter_expression:
        # Otherwise, perform a scan operation
        response = table.scan(FilterExpression=filter_expression)
    else:
        # Return empty response if no values provided
        return {"Items": [], "Count": 0}

    # Handle pagination if there are more results
    items = response.get("Items", [])
    while "LastEvaluatedKey" in response:
        if partition_key_name and partition_key_value:
            response = table.query(
                KeyConditionExpression=key_condition,
                FilterExpression=filter_expression,
                ExclusiveStartKey=response["LastEvaluatedKey"],
            )
        else:
            response = table.scan(
                FilterExpression=filter_expression,
 ExclusiveStartKey=response["LastEvaluatedKey"]
            )
        items.extend(response.get("Items", []))

    # Return the complete result
    return {"Items": items, "Count": len(items)}
```

Ejemplo de uso de la comparación de varios valores con AWS SDK for Python (Boto3).

```
def example_usage():
    """Example of how to use the compare_multiple_values function."""
    # Example parameters
    table_name = "Products"
    attribute_name = "Category"
    values_list = ["Electronics", "Computers", "Accessories"]

    print(f"Searching for products in any of these categories: {values_list}")

    # Using the IN operator (recommended approach)
    print("\nApproach 1: Using the IN operator")
```

```
    response = compare_multiple_values(
        table_name=table_name, attribute_name=attribute_name,
values_list=values_list
    )

    print(f"Found {response['Count']} products in the specified categories")

    # Using multiple OR conditions (alternative approach)
    print("\nApproach 2: Using multiple OR conditions")
    response2 = compare_with_or_conditions(
        table_name=table_name, attribute_name=attribute_name,
values_list=values_list
    )

    print(f"Found {response2['Count']} products in the specified categories")

    # Example with a query operation
    print("\nQuerying a specific manufacturer's products in multiple categories")
    partition_key_name = "Manufacturer"
    partition_key_value = "Acme"

    response3 = compare_multiple_values(
        table_name=table_name,
        attribute_name=attribute_name,
        values_list=values_list,
        partition_key_name=partition_key_name,
        partition_key_value=partition_key_value,
    )

    print(f"Found {response3['Count']} Acme products in the specified
categories")

    # Explain the benefits of using the IN operator
    print("\nBenefits of using the IN operator:")
    print("1. More concise expression compared to multiple OR conditions")
    print("2. Better readability and maintainability")
    print("3. Potentially better performance with large value lists")
    print("4. Simpler code that's less prone to errors")
    print("5. Easier to modify when adding or removing values")
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Python (Boto3).
  - Query
  - Examen

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Actualización condicional de un elemento de DynamoDB con un TTL mediante un AWS SDK

En los siguientes ejemplos de código se muestra cómo actualizar de forma condicional el TTL de un elemento.

Java

SDK para Java 2.x

Actualice el TTL en un elemento de DynamoDB existente en una tabla con una condición.

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import
 software.amazon.awssdk.services.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.Map;
import java.util.Optional;

/**
 * Updates an item in a DynamoDB table with TTL attributes using a conditional
 expression.
```

```
 * This class demonstrates how to conditionally update TTL expiration timestamps.
 */
public class UpdateTTLConditional {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
                sortKey - The name of the sort key. Also known as the range
 attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
 is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
    private static final String UPDATED_AT_ATTR = "updatedAt";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String UPDATE_EXPRESSION = "SET " + UPDATED_AT_ATTR +
"=:c, " + EXPIRE_AT_ATTR + "=:e";
    private static final String CONDITION_EXPRESSION = "attribute_exists(" +
PRIMARY_KEY_ATTR + ")";
    private static final String SUCCESS_MESSAGE = "%s UpdateItem operation with
TTL successful.";
    private static final String CONDITION_FAILED_MESSAGE = "Condition check
failed. Item does not exist.";
    private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
DynamoDB table \"%s\" can't be found.";

    private final DynamoDbClient dynamoDbClient;

    /**
     * Constructs an UpdateTTLConditional with a default DynamoDB client.
     */
    public UpdateTTLConditional() {
        this.dynamoDbClient = null;
    }

    /**
```

```
     * Constructs an UpdateTTLConditional with the specified DynamoDB client.
     *
     * @param dynamoDbClient The DynamoDB client to use
     */
    public UpdateTTLConditional(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }

    /**
     * Main method to demonstrate conditionally updating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new UpdateTTLConditional().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and conditionally update an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);
```

```
        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = Map.of(
            PRIMARY_KEY_ATTR, AttributeValue.builder().s(primaryKey).build(),
            SORT_KEY_ATTR, AttributeValue.builder().s(sortKey).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = Map.of(
            ":c",
AttributeValue.builder().n(String.valueOf(currentTime)).build(),
            ":e",
AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .conditionExpression(CONDITION_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final UpdateItemResponse response = ddb.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return 0;
        } catch (ConditionalCheckFailedException e) {
            System.err.println(CONDITION_FAILED_MESSAGE);
            throw e;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
```

```
}
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Actualice el TTL en un elemento de DynamoDB existente en una tabla con una condición.

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItemConditional = async (tableName, partitionKey, sortKey,
 region = 'us-east-1', newAttribute = 'default-value') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            artist: partitionKey,
            album: sortKey
        }),
        UpdateExpression: "SET newAttribute = :newAttribute",
        ConditionExpression: "expireAt > :expiration",
        ExpressionAttributeValues: marshall({
            ':newAttribute': newAttribute,
            ':expiration': currentTime
        }),
        ReturnValues: "ALL_NEW"
    };

    try {
        const response = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(response.Attributes);
        console.log("Item updated successfully: ", responseData);
```

```
        return responseData;
    } catch (error) {
        if (error.name === "ConditionalCheckFailedException") {
            console.log("Condition check failed: Item's 'expireAt' is expired.");
        } else {
            console.error("Error updating item: ", error);
        }
        throw error;
    }
};

// Example usage (commented out for testing)
// updateItemConditional('your-table-name', 'your-partition-key-value', 'your-
sort-key-value');
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Actualice el TTL en un elemento de DynamoDB existente en una tabla con una condición.

```
from datetime import datetime, timedelta

import boto3
from botocore.exceptions import ClientError


def update_dynamodb_item_ttl(table_name, region, primary_key, sort_key,
 ttl_attribute):
    """
    Updates an existing record in a DynamoDB table with a new or updated TTL
 attribute.

    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :param ttl_attribute: name of the TTL attribute in the target DynamoDB table
    :return:
```

```
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Generate updated TTL in epoch second format
        updated_expiration_time = int((datetime.now() +
 timedelta(days=90)).timestamp())

        # Define the update expression for adding/updating a new attribute
        update_expression = "SET newAttribute = :val1"

        # Define the condition expression for checking if 'expireAt' is not
 expired
        condition_expression = "expireAt > :val2"

        # Define the expression attribute values
        expression_attribute_values = {":val1": ttl_attribute, ":val2":
 updated_expiration_time}

        response = table.update_item(
            Key={"primaryKey": primary_key, "sortKey": sort_key},
            UpdateExpression=update_expression,
            ConditionExpression=condition_expression,
            ExpressionAttributeValues=expression_attribute_values,
        )

        print("Item updated successfully.")
        return response["ResponseMetadata"]["HTTPStatusCode"]  # Ideally a 200 OK
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            print("Condition check failed: Item's 'expireAt' is expired.")
        else:
            print(f"Error updating item: {e}")
    except Exception as e:
        print(f"Error updating item: {e}")


# replace with your values
update_dynamodb_item_ttl(
    "your-table-name",
    "us-east-1",
    "your-partition-key-value",
    "your-sort-key-value",
```

```
    "your-ttl-attribute-value",
)
```

- Para obtener información la API, consulte UpdateItem en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Conexión a una instancia local de DynamoDB mediante un SDK de AWS

El siguiente ejemplo de código muestra cómo invalidar la URL de un punto de conexión para conectarse a una implementación de desarrollo local de DynamoDB y a un SDK de AWS.

Para obtener más información, consulte DynamoDB Local.

Rust

SDK para Rust

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
/// Lists your tables from a local DynamoDB instance by setting the SDK Config's
/// endpoint_url and test_credentials.
#[tokio::main]
async fn main() {
    tracing_subscriber::fmt::init();

    let config = aws_config::defaults(aws_config::BehaviorVersion::latest())
        .test_credentials()
        // DynamoDB run locally uses port 8000 by default.
        .endpoint_url("http://localhost:8000")
        .load()
        .await;
```

```
    let dynamodb_local_config =
 aws_sdk_dynamodb::config::Builder::from(&config).build();

    let client = aws_sdk_dynamodb::Client::from_conf(dynamodb_local_config);

    let list_resp = client.list_tables().send().await;
    match list_resp {
        Ok(resp) => {
            println!("Found {} tables", resp.table_names().len());
            for name in resp.table_names() {
                println!("  {}", name);
            }
        }
        Err(err) => eprintln!("Failed to list local dynamodb tables: {err:?}"),
    }
}
```

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Recuento de operadores de expresión en DynamoDB con un AWS SDK

En los siguientes ejemplos de código se muestra cómo contar operadores de expresión en DynamoDB.

- Conozca el límite de 300 operadores de DynamoDB.
- Cuente los operadores en expresiones complejas.
- Optimice las expresiones para mantenerse dentro de los límites.

Java

SDK para Java 2.x

Demuestre el recuento de operadores de expresión con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
```

```
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

    /**
     * Creates a complex filter expression with a specified number of conditions.
     *
     * <p>This method demonstrates how to generate a complex expression with
     * a specific number of operators to test the 300 operator limit.
     *
     * @param conditionsCount Number of conditions to include
     * @param useAnd Whether to use AND (true) or OR (false) between conditions
     * @return Map containing the filter expression, attribute values, and
 operator count
     */
    public static Map<String, Object> createComplexFilterExpression(int
 conditionsCount, boolean useAnd) {
        // Initialize the expression parts and attribute values
        StringBuilder filterExpression = new StringBuilder();
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();

        // Generate the specified number of conditions
        for (int i = 0; i < conditionsCount; i++) {
            // Add the operator between conditions (except for the first one)
            if (i > 0) {
                filterExpression.append(useAnd ? " AND " : " OR ");
            }

            // Alternate between different comparison operators for variety
            String valueKey = ":val" + i;

            switch (i % 5) {
                case 0:
                    filterExpression.append("attribute").append(i).append(" =
 ").append(valueKey);
                    expressionAttributeValues.put(
                        valueKey, AttributeValue.builder().s("value" +
 i).build());
                    break;
                case 1:
```

```
                    filterExpression.append("attribute").append(i).append(" >
").append(valueKey);
                    expressionAttributeValues.put(
                        valueKey,
AttributeValue.builder().n(String.valueOf(i)).build());
                    break;
                case 2:
                    filterExpression.append("attribute").append(i).append(" <
").append(valueKey);
                    expressionAttributeValues.put(
                        valueKey,
                        AttributeValue.builder().n(String.valueOf(i *
10)).build());
                    break;
                case 3:
                    filterExpression
                        .append("contains(attribute")
                        .append(i)
                        .append(", ")
                        .append(valueKey)
                        .append(")");
                    expressionAttributeValues.put(
                        valueKey, AttributeValue.builder().s("substring" +
i).build());
                    break;
                case 4:
                    filterExpression
                        .append("attribute_exists(attribute")
                        .append(i)
                        .append(")");
                    break;
                default:
                    // This case will never be reached, but added to satisfy
checkstyle
                    break;
            }
        }

        // Calculate the operator count
        // Each condition has 1 operator (=, >, <, contains, attribute_exists)
        // Each AND or OR between conditions is 1 operator
        int operatorCount = conditionsCount + (conditionsCount > 0 ?
conditionsCount - 1 : 0);
```

```
        // Create the result map
        Map<String, Object> result = new HashMap<>();
        result.put("filterExpression", filterExpression.toString());
        result.put("expressionAttributeValues", expressionAttributeValues);
        result.put("operatorCount", operatorCount);

        return result;
    }

    /**
     * Creates a complex update expression with a specified number of operations.
     *
     * <p>This method demonstrates how to generate a complex update expression
with
     * a specific number of operators to test the 300 operator limit.
     *
     * @param operationsCount Number of operations to include
     * @return Map containing the update expression, attribute values, and
operator count
     */
    public static Map<String, Object> createComplexUpdateExpression(int
operationsCount) {
        // Initialize the expression parts and attribute values
        StringBuilder updateExpression = new StringBuilder("SET ");
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();

        // Generate the specified number of SET operations
        for (int i = 0; i < operationsCount; i++) {
            // Add comma between operations (except for the first one)
            if (i > 0) {
                updateExpression.append(", ");
            }

            // Alternate between different types of SET operations
            String valueKey = ":val" + i;

            switch (i % 3) {
                case 0:
                    // Simple assignment (1 operator: =)
                    updateExpression.append("attribute").append(i).append(" =
").append(valueKey);
                    expressionAttributeValues.put(
                        valueKey, AttributeValue.builder().s("value" +
i).build());
```

```
                    break;
                case 1:
                    // Addition (2 operators: = and +)
                    updateExpression
                        .append("attribute")
                        .append(i)
                        .append(" = attribute")
                        .append(i)
                        .append(" + ")
                        .append(valueKey);
                    expressionAttributeValues.put(
                        valueKey,
AttributeValue.builder().n(String.valueOf(i)).build());
                    break;
                case 2:
                    // Conditional assignment with if_not_exists (2 operators: =
and if_not_exists)
                    updateExpression
                        .append("attribute")
                        .append(i)
                        .append(" = if_not_exists(attribute")
                        .append(i)
                        .append(", ")
                        .append(valueKey)
                        .append(")");
                    expressionAttributeValues.put(
                        valueKey,
                        AttributeValue.builder().n(String.valueOf(i *
10)).build());
                    break;
                default:
                    // This case will never be reached, but added to satisfy
checkstyle
                    break;
            }
        }

        // Calculate the operator count
        // Each operation has 1-2 operators as noted above
        int operatorCount = 0;
        for (int i = 0; i < operationsCount; i++) {
            operatorCount += (i % 3 == 0) ? 1 : 2;
        }
```

```
        // Create the result map
        Map<String, Object> result = new HashMap<>();
        result.put("updateExpression", updateExpression.toString());
        result.put("expressionAttributeValues", expressionAttributeValues);
        result.put("operatorCount", operatorCount);

        return result;
    }

    /**
     * Test the operator limit by attempting an operation with a complex
expression.
     *
     * <p>This method demonstrates what happens when an expression approaches or
     * exceeds the 300 operator limit.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param operatorCount Target number of operators to include
     * @return Map containing the result of the operation attempt
     */
    public static Map<String, Object> testOperatorLimit(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, int operatorCount) {

        // Create a complex update expression with the specified operator count
        Map<String, Object> expressionData =
            createComplexUpdateExpression((int) Math.ceil(operatorCount /
1.5)); // Adjust to get close to target count

        String updateExpression = (String)
expressionData.get("updateExpression");
        @SuppressWarnings("unchecked")
        Map<String, AttributeValue> expressionAttributeValues =
            (Map<String, AttributeValue>)
expressionData.get("expressionAttributeValues");
        int actualCount = (int) expressionData.get("operatorCount");

        System.out.println("Generated update expression with approximately " +
actualCount + " operators");

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
```

```
            .tableName(tableName)
            .key(key)
            .updateExpression(updateExpression)
            .expressionAttributeValues(expressionAttributeValues)
            .returnValues("UPDATED_NEW")
            .build();

        try {
            // Attempt the update operation
            UpdateItemResponse response = dynamoDbClient.updateItem(request);

            Map<String, Object> result = new HashMap<>();
            result.put("success", true);
            result.put("message", "Operation succeeded with " + actualCount + "
operators");
            result.put("data", response);
            return result;

        } catch (DynamoDbException e) {
            // Check if the error is due to exceeding the operator limit
            if (e.getMessage().contains("too many operators")) {
                Map<String, Object> result = new HashMap<>();
                result.put("success", false);
                result.put("message", "Operation failed: " + e.getMessage());
                result.put("operatorCount", actualCount);
                return result;
            }

            // Return other errors
            Map<String, Object> result = new HashMap<>();
            result.put("success", false);
            result.put("message", "Operation failed: " + e.getMessage());
            result.put("error", e);
            return result;
        }
    }

    /**
     * Break down a complex expression into multiple simpler operations.
     *
     * <p>This method demonstrates how to handle expressions that would exceed
     * the 300 operator limit by breaking them into multiple operations.
     *
     * @param dynamoDbClient The DynamoDB client
```

```
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param totalOperations Total number of operations to perform
     * @return Map containing the results of the operations
     */
    public static Map<String, Object> breakDownComplexExpression(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, int totalOperations) {

        // Calculate how many operations we can safely include in each batch
        // Using 150 as a conservative limit (well below 300)
        final int operationsPerBatch = 100;
        final int batchCount = (int) Math.ceil((double) totalOperations /
operationsPerBatch);

        System.out.println("Breaking down " + totalOperations + " operations into
" + batchCount + " batches");

        Map<String, Object> results = new HashMap<>();
        results.put("totalBatches", batchCount);

        Map<Integer, Map<String, Object>> batchResults = new HashMap<>();

        // Process each batch
        for (int batch = 0; batch < batchCount; batch++) {
            // Calculate the operations for this batch
            int batchStart = batch * operationsPerBatch;
            int batchEnd = Math.min(batchStart + operationsPerBatch,
totalOperations);
            int batchSize = batchEnd - batchStart;

            System.out.println(
                "Processing batch " + (batch + 1) + "/" + batchCount + " with " +
batchSize + " operations");

            // Create an update expression for this batch
            Map<String, Object> expressionData =
createComplexUpdateExpression(batchSize);

            String updateExpression = (String)
expressionData.get("updateExpression");
            @SuppressWarnings("unchecked")
            Map<String, AttributeValue> expressionAttributeValues =
```

```
                (Map<String, AttributeValue>)
expressionData.get("expressionAttributeValues");
            int operatorCount = (int) expressionData.get("operatorCount");

            // Define the update parameters
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression(updateExpression)
                .expressionAttributeValues(expressionAttributeValues)
                .returnValues("UPDATED_NEW")
                .build();

            try {
                // Perform the update operation for this batch
                UpdateItemResponse response = dynamoDbClient.updateItem(request);

                Map<String, Object> batchResult = new HashMap<>();
                batchResult.put("batch", batch + 1);
                batchResult.put("success", true);
                batchResult.put("operatorCount", operatorCount);
                batchResult.put("attributes", response.attributes());

                batchResults.put(batch, batchResult);

            } catch (DynamoDbException e) {
                Map<String, Object> batchResult = new HashMap<>();
                batchResult.put("batch", batch + 1);
                batchResult.put("success", false);
                batchResult.put("operatorCount", operatorCount);
                batchResult.put("error", e.getMessage());

                batchResults.put(batch, batchResult);

                // Continue with next batch instead of breaking
                continue;
            }
        }

        results.put("results", batchResults);
        return results;
    }

    /**
```

```
     * Count operators in a DynamoDB expression based on the rules in the
 documentation.
     *
     * <p>This method demonstrates how operators are counted according to the
     * DynamoDB documentation.
     *
     * @param expression The DynamoDB expression to analyze
     * @return Map containing the breakdown of operator counts
     */
    public static Map<String, Integer> countOperatorsInExpression(String
 expression) {
        // Initialize counters for different operator types
        Map<String, Integer> counts = new HashMap<>();
        counts.put("comparisonOperators", 0);
        counts.put("logicalOperators", 0);
        counts.put("functions", 0);
        counts.put("arithmeticOperators", 0);
        counts.put("specialOperators", 0);
        counts.put("total", 0);

        // Count comparison operators (=, <>, <, <=, >, >=)
        // This is a simplified approach and may not catch all cases
        int comparisonCount = 0;
        Pattern comparisonPattern = Pattern.compile("(=|<>|<=|>=|<|>)");
        Matcher comparisonMatcher = comparisonPattern.matcher(expression);
        while (comparisonMatcher.find()) {
            comparisonCount++;
        }
        counts.put("comparisonOperators", comparisonCount);

        // Count logical operators (AND, OR, NOT)
        int andCount = countOccurrences(expression, "\\bAND\\b");
        int orCount = countOccurrences(expression, "\\bOR\\b");
        int notCount = countOccurrences(expression, "\\bNOT\\b");
        counts.put("logicalOperators", andCount + orCount + notCount);

        // Count functions (attribute_exists, attribute_not_exists,
 attribute_type, begins_with, contains, size)
        int functionCount = countOccurrences(
            expression,
            "\\b(attribute_exists|attribute_not_exists|attribute_type|
begins_with|contains|size|if_not_exists)\\(");
        counts.put("functions", functionCount);
```

```
        // Count arithmetic operators (+ and -)
        // This is a simplified approach and may not catch all cases
        int arithmeticCount = 0;
        Pattern arithmeticPattern = Pattern.compile("[a-zA-Z0-9_)\\]]\\s*[\\+\
\-]\\s*[a-zA-Z0-9_:(]");
        Matcher arithmeticMatcher = arithmeticPattern.matcher(expression);
        while (arithmeticMatcher.find()) {
            arithmeticCount++;
        }
        counts.put("arithmeticOperators", arithmeticCount);

        // Count special operators (BETWEEN, IN)
        int betweenCount = countOccurrences(expression, "\\bBETWEEN\\b");
        int inCount = countOccurrences(expression, "\\bIN\\b");
        counts.put("specialOperators", betweenCount + inCount);

        // Add extra operators for BETWEEN (each BETWEEN includes an AND)
        int currentLogicalOps = counts.getOrDefault("logicalOperators", 0);
        counts.put("logicalOperators", currentLogicalOps + betweenCount);

        // Calculate total
        int total = counts.getOrDefault("comparisonOperators", 0)
            + counts.getOrDefault("logicalOperators", 0)
            + counts.getOrDefault("functions", 0)
            + counts.getOrDefault("arithmeticOperators", 0)
            + counts.getOrDefault("specialOperators", 0);
        counts.put("total", total);

        return counts;
    }

    /**
     * Helper method to count occurrences of a pattern in a string.
     *
     * @param text The text to search in
     * @param regex The regular expression pattern to search for
     * @return The number of occurrences
     */
    private static int countOccurrences(String text, String regex) {
        final Pattern pattern = Pattern.compile(regex);
        final Matcher matcher = pattern.matcher(text);
        int count = 0;
        while (matcher.find()) {
            count++;
```

```
        }
        return count;
    }
```

Ejemplo de uso del recuento de operadores de expresión con AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
       // Example key
       Map<String, AttributeValue> key = new HashMap<>();
       key.put("ProductId", AttributeValue.builder().s("P12345").build());

       System.out.println("Demonstrating DynamoDB expression operator counting
and the 300 operator limit");

       try {
           // Example 1: Analyze a simple expression
           System.out.println("\nExample 1: Analyzing a simple expression");
           String simpleExpression = "Price = :price AND Rating > :rating AND
Category IN (:cat1, :cat2, :cat3)";
           Map<String, Integer> simpleCount =
countOperatorsInExpression(simpleExpression);

           System.out.println("Expression: " + simpleExpression);
           System.out.println("Operator count breakdown:");
           System.out.println("- Comparison operators: " +
simpleCount.get("comparisonOperators"));
           System.out.println("- Logical operators: " +
simpleCount.get("logicalOperators"));
           System.out.println("- Functions: " + simpleCount.get("functions"));
           System.out.println("- Arithmetic operators: " +
simpleCount.get("arithmeticOperators"));
           System.out.println("- Special operators: " +
simpleCount.get("specialOperators"));
           System.out.println("- Total operators: " + simpleCount.get("total"));

           // Example 2: Analyze a complex expression
           System.out.println("\nExample 2: Analyzing a complex expression");
           String complexExpression = "(attribute_exists(Category) AND Size
BETWEEN :min AND :max) OR "
                   + "(Price > :price AND contains(Description, :keyword) AND "
                   + "(Rating >= :minRating OR Reviews > :minReviews))";
```

```
            Map<String, Integer> complexCount =
countOperatorsInExpression(complexExpression);

            System.out.println("Expression: " + complexExpression);
            System.out.println("Operator count breakdown:");
            System.out.println("- Comparison operators: " +
complexCount.get("comparisonOperators"));
            System.out.println("- Logical operators: " +
complexCount.get("logicalOperators"));
            System.out.println("- Functions: " + complexCount.get("functions"));
            System.out.println("- Arithmetic operators: " +
complexCount.get("arithmeticOperators"));
            System.out.println("- Special operators: " +
complexCount.get("specialOperators"));
            System.out.println("- Total operators: " +
complexCount.get("total"));

            // Example 3: Test approaching the operator limit
            System.out.println("\nExample 3: Testing an expression approaching
the operator limit");
            Map<String, Object> approachingLimit =
testOperatorLimit(dynamoDbClient, tableName, key, 290);
            System.out.println(approachingLimit.get("message"));

            // Example 4: Test exceeding the operator limit
            System.out.println("\nExample 4: Testing an expression exceeding the
operator limit");
            Map<String, Object> exceedingLimit =
testOperatorLimit(dynamoDbClient, tableName, key, 310);
            System.out.println(exceedingLimit.get("message"));

            // Example 5: Breaking down a complex expression
            System.out.println("\nExample 5: Breaking down a complex expression
into multiple operations");
            Map<String, Object> breakdownResult =
breakDownComplexExpression(dynamoDbClient, tableName, key, 500);
            @SuppressWarnings("unchecked")
            Map<Integer, Map<String, Object>> results =
                (Map<Integer, Map<String, Object>>)
breakdownResult.get("results");
            System.out.println(
                "Processed " + results.size() + " of " +
breakdownResult.get("totalBatches") + " batches");
```

```
            // Explain the operator counting rules
            System.out.println("\nKey points about DynamoDB expression operator
counting:");
            System.out.println("1. The maximum number of operators in any
expression is 300");
            System.out.println("2. Each comparison operator (=, <>, <, <=, >, >=)
counts as 1 operator");
            System.out.println("3. Each logical operator (AND, OR, NOT) counts as
1 operator");
            System.out.println("4. Each function call (attribute_exists,
contains, etc.) counts as 1 operator");
            System.out.println("5. Each arithmetic operator (+ or -) counts as 1
operator");
            System.out.println("6. BETWEEN counts as 2 operators (BETWEEN itself
and the AND within it)");
            System.out.println("7. IN counts as 1 operator regardless of the
number of values");
            System.out.println("8. Parentheses for grouping and attribute paths
don't count as operators");
            System.out.println("9. When you exceed the limit, the error always
reports '301 operators'");
            System.out.println("10. For complex operations, break them into
multiple smaller operations");

        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Demuestre el recuento de operadores de expresión con AWS SDK para JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
```

```
  UpdateCommand,
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Create a complex filter expression with a specified number of conditions.
 *
 * This function demonstrates how to generate a complex expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} conditionsCount - Number of conditions to include
 * @param {boolean} useAnd - Whether to use AND (true) or OR (false) between
 conditions
 * @returns {Object} - Object containing the filter expression and attribute
 values
 */
function createComplexFilterExpression(conditionsCount, useAnd = true) {
  // Initialize the expression parts and attribute values
  const conditions = [];
  const expressionAttributeValues = {};

  // Generate the specified number of conditions
  for (let i = 0; i < conditionsCount; i++) {
    // Alternate between different comparison operators for variety
    let condition;
    const valueKey = `:val${i}`;

    switch (i % 5) {
      case 0:
        condition = `attribute${i} = ${valueKey}`;
        expressionAttributeValues[valueKey] = `value${i}`;
        break;
      case 1:
        condition = `attribute${i} > ${valueKey}`;
        expressionAttributeValues[valueKey] = i;
        break;
      case 2:
        condition = `attribute${i} < ${valueKey}`;
        expressionAttributeValues[valueKey] = i * 10;
        break;
      case 3:
        condition = `contains(attribute${i}, ${valueKey})`;
        expressionAttributeValues[valueKey] = `substring${i}`;
        break;
```

```
      case 4:
        condition = `attribute_exists(attribute${i})`;
        break;
    }

    conditions.push(condition);
  }

  // Join the conditions with AND or OR
  const operator = useAnd ? " AND " : " OR ";
  const filterExpression = conditions.join(operator);

  // Calculate the operator count
  // Each condition has 1 operator (=, >, <, contains, attribute_exists)
  // Each AND or OR between conditions is 1 operator
  const operatorCount = conditionsCount + (conditionsCount > 0 ? conditionsCount
 - 1 : 0);

  return {
    filterExpression,
    expressionAttributeValues,
    operatorCount
  };
}

/**
 * Create a complex update expression with a specified number of operations.
 *
 * This function demonstrates how to generate a complex update expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} operationsCount - Number of operations to include
 * @returns {Object} - Object containing the update expression and attribute
 values
 */
function createComplexUpdateExpression(operationsCount) {
  // Initialize the expression parts and attribute values
  const setOperations = [];
  const expressionAttributeValues = {};

  // Generate the specified number of SET operations
  for (let i = 0; i < operationsCount; i++) {
    // Alternate between different types of SET operations
    let operation;
```

```
    const valueKey = `:val${i}`;

    switch (i % 3) {
      case 0:
        // Simple assignment (1 operator: =)
        operation = `attribute${i} = ${valueKey}`;
        expressionAttributeValues[valueKey] = `value${i}`;
        break;
      case 1:
        // Addition (2 operators: = and +)
        operation = `attribute${i} = attribute${i} + ${valueKey}`;
        expressionAttributeValues[valueKey] = i;
        break;
      case 2:
        // Conditional assignment with if_not_exists (2 operators: = and
 if_not_exists)
        operation = `attribute${i} = if_not_exists(attribute${i}, ${valueKey})`;
        expressionAttributeValues[valueKey] = i * 10;
        break;
    }

    setOperations.push(operation);
  }

  // Create the update expression
  const updateExpression = `SET ${setOperations.join(", ")}`;

  // Calculate the operator count
  // Each operation has 1-2 operators as noted above
  let operatorCount = 0;
  for (let i = 0; i < operationsCount; i++) {
    operatorCount += (i % 3 === 0) ? 1 : 2;
  }

  return {
    updateExpression,
    expressionAttributeValues,
    operatorCount
  };
}

/**
 * Test the operator limit by attempting an operation with a complex expression.
 *
```

```
 * This function demonstrates what happens when an expression approaches or
 * exceeds the 300 operator limit.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} operatorCount - Target number of operators to include
 * @returns {Promise<Object>} - Result of the operation attempt
 */
async function testOperatorLimit(
  config,
  tableName,
  key,
  operatorCount
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create a complex update expression with the specified operator count
  const { updateExpression, expressionAttributeValues, operatorCount:
 actualCount } =
    createComplexUpdateExpression(Math.ceil(operatorCount / 1.5)); // Adjust to
 get close to target count

  console.log(`Generated update expression with approximately ${actualCount}
 operators`);

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Attempt the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return {
      success: true,
      message: `Operation succeeded with ${actualCount} operators`,
      data: response
```

```
    };
  } catch (error) {
    // Check if the error is due to exceeding the operator limit
    if (error.name === "ValidationException" &&
        error.message.includes("too many operators")) {
      return {
        success: false,
        message: `Operation failed: ${error.message}`,
        operatorCount: actualCount
      };
    }

    // Return other errors
    return {
      success: false,
      message: `Operation failed: ${error.message}`,
      error
    };
  }
}

/**
 * Break down a complex expression into multiple simpler operations.
 *
 * This function demonstrates how to handle expressions that would exceed
 * the 300 operator limit by breaking them into multiple operations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} totalOperations - Total number of operations to perform
 * @returns {Promise<Object>} - Result of the operations
 */
async function breakDownComplexExpression(
  config,
  tableName,
  key,
  totalOperations
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Calculate how many operations we can safely include in each batch
```

```
 // Using 150 as a conservative limit (well below 300)
 const operationsPerBatch = 100;
 const batchCount = Math.ceil(totalOperations / operationsPerBatch);

 console.log(`Breaking down ${totalOperations} operations into ${batchCount}
batches`);

 const results = [];

 // Process each batch
 for (let batch = 0; batch < batchCount; batch++) {
   // Calculate the operations for this batch
   const batchStart = batch * operationsPerBatch;
   const batchEnd = Math.min(batchStart + operationsPerBatch, totalOperations);
   const batchSize = batchEnd - batchStart;

   console.log(`Processing batch ${batch + 1}/${batchCount} with ${batchSize}
operations`);

   // Create an update expression for this batch
   const { updateExpression, expressionAttributeValues, operatorCount } =
     createComplexUpdateExpression(batchSize);

   // Define the update parameters
   const params = {
     TableName: tableName,
     Key: key,
     UpdateExpression: updateExpression,
     ExpressionAttributeValues: expressionAttributeValues,
     ReturnValues: "UPDATED_NEW"
   };

   try {
     // Perform the update operation for this batch
     const response = await docClient.send(new UpdateCommand(params));

     results.push({
       batch: batch + 1,
       success: true,
       operatorCount,
       attributes: response.Attributes
     });
   } catch (error) {
     results.push({
```

```
        batch: batch + 1,
        success: false,
        operatorCount,
        error: error.message
      });

      // Stop processing if an error occurs
      break;
    }
  }

  return {
    totalBatches: batchCount,
    results
  };
}

/**
 * Count operators in a DynamoDB expression based on the rules in the
 documentation.
 *
 * This function demonstrates how operators are counted according to the
 * DynamoDB documentation.
 *
 * @param {string} expression - The DynamoDB expression to analyze
 * @returns {Object} - Breakdown of operator counts
 */
function countOperatorsInExpression(expression) {
  // Initialize counters for different operator types
  const counts = {
    comparisonOperators: 0,
    logicalOperators: 0,
    functions: 0,
    arithmeticOperators: 0,
    specialOperators: 0,
    total: 0
  };

  // Count comparison operators (=, <>, <, <=, >, >=)
  const comparisonRegex = /[^<>]=[^=]|<>|<=|>=|[^<]>[^=]|[^>]<[^=]/g;
  const comparisonMatches = expression.match(comparisonRegex) || [];
  counts.comparisonOperators = comparisonMatches.length;

  // Count logical operators (AND, OR, NOT)
```

```
  const andMatches = expression.match(/\bAND\b/g) || [];
  const orMatches = expression.match(/\bOR\b/g) || [];
  const notMatches = expression.match(/\bNOT\b/g) || [];
  counts.logicalOperators = andMatches.length + orMatches.length +
 notMatches.length;

  // Count functions (attribute_exists, attribute_not_exists, attribute_type,
 begins_with, contains, size)
  const functionRegex = /\b(attribute_exists|attribute_not_exists|attribute_type|
begins_with|contains|size|if_not_exists)\(/g;
  const functionMatches = expression.match(functionRegex) || [];
  counts.functions = functionMatches.length;

  // Count arithmetic operators (+ and -)
  const arithmeticMatches = expression.match(/[a-zA-Z0-9_)\]]\s*[\+\-]\s*[a-zA-
Z0-9_(:]/g) || [];
  counts.arithmeticOperators = arithmeticMatches.length;

  // Count special operators (BETWEEN, IN)
  const betweenMatches = expression.match(/\bBETWEEN\b/g) || [];
  const inMatches = expression.match(/\bIN\b/g) || [];
  counts.specialOperators = betweenMatches.length + inMatches.length;

  // Add extra operators for BETWEEN (each BETWEEN includes an AND)
  counts.logicalOperators += betweenMatches.length;

  // Calculate total
  counts.total = counts.comparisonOperators +
                 counts.logicalOperators +
                 counts.functions +
                 counts.arithmeticOperators +
                 counts.specialOperators;

  return counts;
}
```

Ejemplo de uso del recuento de operadores de expresión con AWS SDK para JavaScript.

```
/**
 * Example of how to work with expression operator counting.
 */
async function exampleUsage() {
```

```
 // Example parameters
 const config = { region: "us-west-2" };
 const tableName = "Products";
 const key = { ProductId: "P12345" };

 console.log("Demonstrating DynamoDB expression operator counting and the 300
operator limit");

 try {
   // Example 1: Analyze a simple expression
   console.log("\nExample 1: Analyzing a simple expression");
   const simpleExpression = "Price = :price AND Rating > :rating AND Category IN
(:cat1, :cat2, :cat3)";
   const simpleCount = countOperatorsInExpression(simpleExpression);

   console.log(`Expression: ${simpleExpression}`);
   console.log("Operator count breakdown:");
   console.log(`- Comparison operators: ${simpleCount.comparisonOperators}`);
   console.log(`- Logical operators: ${simpleCount.logicalOperators}`);
   console.log(`- Functions: ${simpleCount.functions}`);
   console.log(`- Arithmetic operators: ${simpleCount.arithmeticOperators}`);
   console.log(`- Special operators: ${simpleCount.specialOperators}`);
   console.log(`- Total operators: ${simpleCount.total}`);

   // Example 2: Analyze a complex expression
   console.log("\nExample 2: Analyzing a complex expression");
   const complexExpression =
     "(attribute_exists(Category) AND Size BETWEEN :min AND :max) OR " +
     "(Price > :price AND contains(Description, :keyword) AND " +
     "(Rating >= :minRating OR Reviews > :minReviews))";
   const complexCount = countOperatorsInExpression(complexExpression);

   console.log(`Expression: ${complexExpression}`);
   console.log("Operator count breakdown:");
   console.log(`- Comparison operators: ${complexCount.comparisonOperators}`);
   console.log(`- Logical operators: ${complexCount.logicalOperators}`);
   console.log(`- Functions: ${complexCount.functions}`);
   console.log(`- Arithmetic operators: ${complexCount.arithmeticOperators}`);
   console.log(`- Special operators: ${complexCount.specialOperators}`);
   console.log(`- Total operators: ${complexCount.total}`);

   // Example 3: Test approaching the operator limit
   console.log("\nExample 3: Testing an expression approaching the operator
limit");
```

```
    const approachingLimit = await testOperatorLimit(config, tableName, key,
 290);
    console.log(approachingLimit.message);

    // Example 4: Test exceeding the operator limit
    console.log("\nExample 4: Testing an expression exceeding the operator
 limit");
    const exceedingLimit = await testOperatorLimit(config, tableName, key, 310);
    console.log(exceedingLimit.message);

    // Example 5: Breaking down a complex expression
    console.log("\nExample 5: Breaking down a complex expression into multiple
 operations");
    const breakdownResult = await breakDownComplexExpression(config, tableName,
 key, 500);
    console.log(`Processed ${breakdownResult.results.length} of
 ${breakdownResult.totalBatches} batches`);

    // Explain the operator counting rules
    console.log("\nKey points about DynamoDB expression operator counting:");
    console.log("1. The maximum number of operators in any expression is 300");
    console.log("2. Each comparison operator (=, <>, <, <=, >, >=) counts as 1
 operator");
    console.log("3. Each logical operator (AND, OR, NOT) counts as 1 operator");
    console.log("4. Each function call (attribute_exists, contains, etc.) counts
 as 1 operator");
    console.log("5. Each arithmetic operator (+ or -) counts as 1 operator");
    console.log("6. BETWEEN counts as 2 operators (BETWEEN itself and the AND
 within it)");
    console.log("7. IN counts as 1 operator regardless of the number of values");
    console.log("8. Parentheses for grouping and attribute paths don't count as
 operators");
    console.log("9. When you exceed the limit, the error always reports '301
 operators'");
    console.log("10. For complex operations, break them into multiple smaller
 operations");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Demuestre el recuento de operadores de expresión con AWS SDK for Python (Boto3).

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, List, Optional, Tuple



def create_complex_filter_expression(
    attribute_name: str, values: List[Any], use_or: bool = True
) -> Tuple[str, Dict[str, Any], Dict[str, str], int]:
    """
    Create a complex filter expression with multiple conditions.

    This function demonstrates how to build a complex filter expression
    and count the number of operators used.

    Args:
        attribute_name (str): The name of the attribute to filter on.
        values (List[Any]): List of values to compare against.
        use_or (bool, optional): Whether to use OR between conditions. Defaults
 to True.

    Returns:
        Tuple[str, Dict[str, Any], Dict[str, str], int]: A tuple containing:
            - The filter expression string
            - Expression attribute values
            - Expression attribute names
            - The number of operators used
    """
    if not values:
        return "", {}, {}, 0

    # Initialize expression components
    filter_expression = ""
    expression_attribute_values = {}
```

```
    expression_attribute_names = {"#attr": attribute_name}
    operator_count = 0

    # Build the filter expression
    for i, value in enumerate(values):
        value_placeholder = f":val{i}"
        expression_attribute_values[value_placeholder] = value

        if i > 0:
            # Add OR or AND operator between conditions
            filter_expression += " OR " if use_or else " AND "
            operator_count += 1  # Count the OR/AND operator

        # Add the condition
        filter_expression += f"#attr = {value_placeholder}"
        operator_count += 1  # Count the = operator

    return (
        filter_expression,
        expression_attribute_values,
        expression_attribute_names,
        operator_count,
    )


def create_nested_filter_expression(
    depth: int, conditions_per_level: int
) -> Tuple[str, Dict[str, Any], Dict[str, str], int]:
    """
    Create a deeply nested filter expression with multiple conditions.

    This function demonstrates how to build a complex nested filter expression
    and count the number of operators used.

    Args:
        depth (int): The depth of nesting.
        conditions_per_level (int): Number of conditions at each level.

    Returns:
        Tuple[str, Dict[str, Any], Dict[str, str], int]: A tuple containing:
            - The filter expression string
            - Expression attribute values
            - Expression attribute names
            - The number of operators used
```

```
    """
    if depth <= 0 or conditions_per_level <= 0:
        return "", {}, {}, 0

    # Initialize expression components
    expression_attribute_values = {}
    expression_attribute_names = {}
    operator_count = 0

    def build_nested_expression(current_depth: int, prefix: str) -> str:
        nonlocal operator_count

        if current_depth <= 0:
            return ""

        # Build conditions at this level
        conditions = []
        for i in range(conditions_per_level):
            attr_name = f"attr{prefix}_{i}"
            attr_placeholder = f"#attr{prefix}_{i}"
            val_placeholder = f":val{prefix}_{i}"

            expression_attribute_names[attr_placeholder] = attr_name
            expression_attribute_values[val_placeholder] = i

            conditions.append(f"{attr_placeholder} = {val_placeholder}")
            operator_count += 1  # Count the = operator

        # Join conditions with AND
        level_expression = " AND ".join(conditions)
        operator_count += max(0, len(conditions) - 1)  # Count the AND operators

        # If not at the deepest level, add nested expressions
        if current_depth > 1:
            nested_expr = build_nested_expression(current_depth - 1,
f"{prefix}_{current_depth}")
            if nested_expr:
                level_expression = f"({level_expression}) OR ({nested_expr})"
                operator_count += 1  # Count the OR operator

        return level_expression

    # Build the expression starting from the top level
    filter_expression = build_nested_expression(depth, "1")
```

```
    return (
        filter_expression,
        expression_attribute_values,
        expression_attribute_names,
        operator_count,
    )


def count_operators_in_update_expression(update_expression: str) -> int:
    """
    Count the number of operators in an update expression.

    This function demonstrates how to count operators in an update expression
    based on DynamoDB's rules.

    Args:
        update_expression (str): The update expression to analyze.

    Returns:
        int: The number of operators in the expression.
    """
    operator_count = 0

    # Count SET operations
    if "SET" in update_expression:
        set_section = (
            update_expression.split("SET")[1].split("REMOVE")[0].split("ADD")
[0].split("DELETE")[0]
        )

        # Count assignment operators (=)
        operator_count += set_section.count("=")

        # Count arithmetic operators (+, -)
        operator_count += set_section.count("+")
        operator_count += set_section.count("-")

        # Count list_append function calls (each counts as 1 operator)
        operator_count += set_section.lower().count("list_append")

        # Count if_not_exists function calls (each counts as 1 operator)
        operator_count += set_section.lower().count("if_not_exists")
```

```
    # Count REMOVE operations (no additional operators)

    # Count ADD operations (each ADD counts as 1 operator)
    if "ADD" in update_expression:
        add_section = (
            update_expression.split("ADD")[1].split("DELETE")[0].split("SET")
[0].split("REMOVE")[0]
        )
        operator_count += add_section.count(",") + 1

    # Count DELETE operations (each DELETE counts as 1 operator)
    if "DELETE" in update_expression:
        delete_section = (
            update_expression.split("DELETE")[1].split("SET")[0].split("ADD")
[0].split("REMOVE")[0]
        )
        operator_count += delete_section.count(",") + 1

    return operator_count


def count_operators_in_condition_expression(condition_expression: str) -> int:
    """
    Count the number of operators in a condition expression.

    This function demonstrates how to count operators in a condition expression
    based on DynamoDB's rules.

    Args:
        condition_expression (str): The condition expression to analyze.

    Returns:
        int: The number of operators in the expression.
    """
    operator_count = 0

    # Count comparison operators
    comparison_operators = ["=", "<>", "<", "<=", ">", ">="]
    for op in comparison_operators:
        operator_count += condition_expression.count(op)

    # Count logical operators
    operator_count += condition_expression.upper().count(" AND ")
    operator_count += condition_expression.upper().count(" OR ")
```

```
    operator_count += condition_expression.upper().count("NOT ")

    # Count BETWEEN operator (counts as 2: BETWEEN + AND)
    between_count = condition_expression.upper().count(" BETWEEN ")
    operator_count += between_count * 2

    # Count IN operator (counts as 1 regardless of number of values)
    operator_count += condition_expression.upper().count(" IN ")

    # Count functions (each counts as 1 operator)
    functions = [
        "attribute_exists",
        "attribute_not_exists",
        "attribute_type",
        "begins_with",
        "contains",
        "size",
    ]
    for func in functions:
        operator_count += condition_expression.lower().count(func)

    return operator_count


# Note: This function is for demonstration purposes only and should be called
 from example_usage()
# It's not meant to be used directly as a test function
def _test_expression_limit(
    table_name: str, key: Dict[str, Any], operator_count: int, attribute_name:
 str = "TestAttribute"
) -> Tuple[bool, Optional[str]]:
    """
    Test if an expression with a specific number of operators exceeds the limit.

    This function demonstrates how to test the 300 operator limit by creating
    an expression with a specified number of operators.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        operator_count (int): The number of operators to include in the
 expression.
        attribute_name (str, optional): The name of the attribute to update.
 Defaults to "TestAttribute".
```

```
    Returns:
        Tuple[bool, Optional[str]]: A tuple containing:
            - A boolean indicating if the operation succeeded
            - The error message if it failed, None otherwise
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Create an update expression with the specified number of operators
    update_expression = f"SET #{attribute_name} = :val0"
    expression_attribute_names = {f"#{attribute_name}": attribute_name}
    expression_attribute_values = {":val0": 0}

    # Add additional SET operations to reach the desired operator count
    # Each assignment adds 1 operator
    for i in range(1, operator_count):
        attr_name = f"{attribute_name}{i}"
        attr_placeholder = f"#attr{i}"
        val_placeholder = f":val{i}"

        update_expression += f", {attr_placeholder} = {val_placeholder}"
        expression_attribute_names[attr_placeholder] = attr_name
        expression_attribute_values[val_placeholder] = i

    try:
        # Attempt the update operation
        table.update_item(
            Key=key,
            UpdateExpression=update_expression,
            ExpressionAttributeNames=expression_attribute_names,
            ExpressionAttributeValues=expression_attribute_values,
        )
        return True, None
    except ClientError as e:
        error_message = e.response["Error"]["Message"]

        if "expression contains too many operators" in error_message.lower():
            return False, error_message
        else:
            # Other error occurred
            raise
```

Ejemplo de uso del recuento de operadores de expresión con AWS SDK for Python (Boto3).

```
def example_usage():
    """Example of how to use the expression operator counting functions."""

    print("Example 1: Creating a complex filter expression with multiple
 conditions")
    attribute_name = "Status"
    values = ["Active", "Pending", "Processing", "Shipped", "Delivered"]

    filter_expr, expr_attr_vals, expr_attr_names, op_count =
 create_complex_filter_expression(
        attribute_name=attribute_name, values=values, use_or=True
    )

    print(f"Filter Expression: {filter_expr}")
    print(f"Expression Attribute Values: {expr_attr_vals}")
    print(f"Expression Attribute Names: {expr_attr_names}")
    print(f"Operator Count: {op_count}")

    print("\nExample 2: Creating a nested filter expression")
    nested_expr, nested_vals, nested_names, nested_count =
 create_nested_filter_expression(
        depth=3, conditions_per_level=2
    )

    print(f"Nested Filter Expression: {nested_expr}")
    print(f"Operator Count: {nested_count}")

    print("\nExample 3: Counting operators in an update expression")
    update_expression = "SET #name = :name, #age = :age + :increment,
 #address.#city = :city, #status = if_not_exists(#status, :default_status) REMOVE
 #old_field ADD #counter :value DELETE #set_attr :set_val"
    update_op_count = count_operators_in_update_expression(update_expression)

    print(f"Update Expression: {update_expression}")
    print(f"Operator Count: {update_op_count}")

    print("\nExample 4: Counting operators in a condition expression")
```

```
    condition_expression = "(#status = :active OR #status = :pending) AND #price
BETWEEN :min_price AND :max_price AND attribute_exists(#category) AND NOT
(#stock <= :min_stock)"
    condition_op_count =
count_operators_in_condition_expression(condition_expression)

    print(f"Condition Expression: {condition_expression}")
    print(f"Operator Count: {condition_op_count}")

    print("\nExample 5: Testing the 300 operator limit")

    # This is just for demonstration - in a real application, you would use your
actual table
    # Note: This function is renamed to _test_expression_limit to avoid pytest
trying to run it
    print("In a real application, you would test with _test_expression_limit
function")
    print("Expression with 250 operators would be under the limit")
    print("Expression with 350 operators would exceed the 300 operator limit")

    print("\nOperator Counting Rules in DynamoDB:")
    print("1. Comparison Operators (=, <>, <, <=, >, >=): 1 operator each")
    print("2. Logical Operators (AND, OR, NOT): 1 operator each")
    print("3. BETWEEN: 2 operators (BETWEEN + AND)")
    print("4. IN: 1 operator (regardless of number of values)")
    print("5. Functions (attribute_exists, begins_with, etc.): 1 operator each")
    print("6. Arithmetic Operators (+, -): 1 operator each")
    print("7. SET assignments (=): 1 operator each")
    print("8. ADD and DELETE operations: 1 operator each")

    print("\nStrategies for Working Within the 300 Operator Limit:")
    print("1. Break operations into multiple requests")
    print("2. Use DynamoDB Transactions for complex operations")
    print("3. Optimize data model to reduce query complexity")
    print("4. Use application-side filtering for less critical filters")
    print("5. Consider using IN operator instead of multiple OR conditions")
```

- Para obtener información la API, consulte UpdateItem en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Creación de una API REST de API Gateway para realizar un seguimiento de datos de COVID-19

En el siguiente ejemplo se muestra cómo crear una API REST que simule un sistema de seguimiento de los casos diarios de COVID-19 en Estados Unidos, con datos ficticios.

Python

SDK para Python (Boto3)

Muestra cómo utilizar AWS Chalice con AWS SDK for Python (Boto3) para crear una API REST sin servidor que utilice Amazon API Gateway, AWS Lambda y Amazon DynamoDB. La API REST simula un sistema que hace el seguimiento de los casos diarios de COVID-19 en Estados Unidos, con datos ficticios. Aprenda cómo:

- Utilizar AWS Chalice para definir rutas en las funciones de Lambda que se llaman para gestionar las solicitudes REST que llegan a través de API Gateway.
- Utilizar funciones de Lambda para recuperar y almacenar datos en una tabla de DynamoDB para atender solicitudes REST.
- Definir la estructura de tabla y los recursos del rol de seguridad en una plantilla de AWS CloudFormation.
- Utilizar AWS Chalice y CloudFormation para empaquetar e implementar todos los recursos necesarios.
- Utilizar CloudFormation para limpiar todos los recursos creados.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- API Gateway
- CloudFormation
- DynamoDB
- Lambda

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Creación de una aplicación de mensajería con Step Functions

En el siguiente ejemplo se muestra cómo crear una aplicación de mensajería de AWS Step Functions que recupere registros de mensajes de una tabla de base de datos.

Python

SDK para Python (Boto3)

Muestra cómo utilizar AWS SDK for Python (Boto3) con AWS Step Functions para crear una aplicación de mensajería que recupere registros de mensajes de una tabla de Amazon DynamoDB y los envíe con Amazon Simple Queue Service (Amazon SQS). La máquina de estado se integra con una función de AWS Lambda para examinar la base de datos en busca de mensajes no enviados.

- Crear una máquina de estado que recupere y actualice los registros de mensajes de una tabla de Amazon DynamoDB.
- Actualizar la definición de la máquina de estado para que también envíe mensajes a Amazon Simple Queue Service (Amazon SQS).
- Iniciar y detener las ejecuciones de la máquina de estado.
- Conectar con Lambda, DynamoDB y Amazon SQS desde una máquina de estado mediante integraciones de servicio.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- DynamoDB
- Lambda
- Amazon SQS
- Step Functions

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Creación de una aplicación de administración de activos fotográficos que permita a los usuarios administrar las fotos mediante etiquetas

En los siguientes ejemplos de código se muestra cómo crear una aplicación sin servidor que permita a los usuarios administrar fotos mediante etiquetas.

.NET

SDK para .NET

Muestra cómo desarrollar una aplicación de administración de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Para profundizar en el origen de este ejemplo, consulte la publicación en Comunidad de AWS.

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

SDK para C++

Muestra cómo desarrollar una aplicación de administración de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Para profundizar en el origen de este ejemplo, consulte la publicación en Comunidad de AWS.

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

SDK para Java 2.x

Muestra cómo desarrollar una aplicación de administración de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Para profundizar en el origen de este ejemplo, consulte la publicación en Comunidad de AWS.

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

SDK para JavaScript (v3)

Muestra cómo desarrollar una aplicación de administración de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Para profundizar en el origen de este ejemplo, consulte la publicación en Comunidad de AWS.

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Kotlin

SDK para Kotlin

Muestra cómo desarrollar una aplicación de administración de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Para profundizar en el origen de este ejemplo, consulte la publicación en Comunidad de AWS.

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda

- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

SDK para PHP

Muestra cómo desarrollar una aplicación de administración de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Para profundizar en el origen de este ejemplo, consulte la publicación en Comunidad de AWS.

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

SDK para Rust

Muestra cómo desarrollar una aplicación de administración de activos fotográficos que detecte las etiquetas de las imágenes mediante Amazon Rekognition y las almacene para su posterior recuperación.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Para profundizar en el origen de este ejemplo, consulte la publicación en Comunidad de AWS.

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Creación de una tabla de DynamoDB con un índice secundario global mediante el AWS SDK

En el siguiente ejemplo de código se muestra cómo crear una tabla con un índice secundario global.

Java

SDK para Java 2.x

Cree una tabla de DynamoDB con un índice secundario global con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableRequest;
import software.amazon.awssdk.services.dynamodb.model.DescribeTableResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndex;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.Projection;
import software.amazon.awssdk.services.dynamodb.model.ProjectionType;
```

```
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.waiters.DynamoDbWaiter;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    public void createTable() {
        try {
            // Attribute definitions
            final List<AttributeDefinition> attributeDefinitions = new
 ArrayList<>();
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(ISSUE_ID_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(TITLE_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(CREATE_DATE_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());
            attributeDefinitions.add(AttributeDefinition.builder()
                .attributeName(DUE_DATE_ATTR)
                .attributeType(ScalarAttributeType.S)
                .build());

            // Key schema for table
            final List<KeySchemaElement> tableKeySchema = new ArrayList<>();
            tableKeySchema.add(KeySchemaElement.builder()
                .attributeName(ISSUE_ID_ATTR)
                .keyType(KeyType.HASH)
                .build()); // Partition key
            tableKeySchema.add(KeySchemaElement.builder()
                .attributeName(TITLE_ATTR)
                .keyType(KeyType.RANGE)
                .build()); // Sort key
```

```
            // Initial provisioned throughput settings for the indexes
            final ProvisionedThroughput ptIndex = ProvisionedThroughput.builder()
                .readCapacityUnits(1L)
                .writeCapacityUnits(1L)
                .build();

            // CreateDateIndex
            final List<KeySchemaElement> createDateKeySchema = new ArrayList<>();
            createDateKeySchema.add(KeySchemaElement.builder()
                .attributeName(CREATE_DATE_ATTR)
                .keyType(KeyType.HASH)
                .build());
            createDateKeySchema.add(KeySchemaElement.builder()
                .attributeName(ISSUE_ID_ATTR)
                .keyType(KeyType.RANGE)
                .build());

            final Projection createDateProjection = Projection.builder()
                .projectionType(ProjectionType.INCLUDE)
                .nonKeyAttributes(DESCRIPTION_ATTR, STATUS_ATTR)
                .build();

            final GlobalSecondaryIndex createDateIndex =
GlobalSecondaryIndex.builder()
                .indexName(CREATE_DATE_INDEX)
                .keySchema(createDateKeySchema)
                .projection(createDateProjection)
                .provisionedThroughput(ptIndex)
                .build();

            // TitleIndex
            final List<KeySchemaElement> titleKeySchema = new ArrayList<>();
            titleKeySchema.add(KeySchemaElement.builder()
                .attributeName(TITLE_ATTR)
                .keyType(KeyType.HASH)
                .build());
            titleKeySchema.add(KeySchemaElement.builder()
                .attributeName(ISSUE_ID_ATTR)
                .keyType(KeyType.RANGE)
                .build());

            final Projection titleProjection =
```

```
Projection.builder().projectionType(ProjectionType.KEYS_ONLY).build();

            final GlobalSecondaryIndex titleIndex =
GlobalSecondaryIndex.builder()
                .indexName(TITLE_INDEX)
                .keySchema(titleKeySchema)
                .projection(titleProjection)
                .provisionedThroughput(ptIndex)
                .build();

            // DueDateIndex
            final List<KeySchemaElement> dueDateKeySchema = new ArrayList<>();
            dueDateKeySchema.add(KeySchemaElement.builder()
                .attributeName(DUE_DATE_ATTR)
                .keyType(KeyType.HASH)
                .build());

            final Projection dueDateProjection =
                Projection.builder().projectionType(ProjectionType.ALL).build();

            final GlobalSecondaryIndex dueDateIndex =
GlobalSecondaryIndex.builder()
                .indexName(DUE_DATE_INDEX)
                .keySchema(dueDateKeySchema)
                .projection(dueDateProjection)
                .provisionedThroughput(ptIndex)
                .build();

            final CreateTableRequest createTableRequest =
CreateTableRequest.builder()
                .tableName(TABLE_NAME)
                .keySchema(tableKeySchema)
                .attributeDefinitions(attributeDefinitions)
                .globalSecondaryIndexes(createDateIndex, titleIndex,
dueDateIndex)
                .provisionedThroughput(ProvisionedThroughput.builder()
                    .readCapacityUnits(1L)
                    .writeCapacityUnits(1L)
                    .build())
                .build();

            System.out.println("Creating table " + TABLE_NAME + "...");
            dynamoDbClient.createTable(createTableRequest);
```

```
            // Wait for table to become active
            System.out.println("Waiting for " + TABLE_NAME + " to become
ACTIVE...");
            final DynamoDbWaiter waiter = dynamoDbClient.waiter();
            final DescribeTableRequest describeTableRequest =
                DescribeTableRequest.builder().tableName(TABLE_NAME).build();

            final WaiterResponse<DescribeTableResponse> waiterResponse =
                waiter.waitUntilTableExists(describeTableRequest);
            waiterResponse.matched().response().ifPresent(response ->
System.out.println("Table is now ready for use"));

        } catch (DynamoDbException e) {
            System.err.println("Error creating table: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Para obtener información sobre la API, consulte CreateTable en la Referencia de la API de AWS SDK for Java 2.x.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Creación de una tabla de DynamoDB con configuración de rendimiento en caliente utilizando un AWS SDK

En los siguientes ejemplos de código se muestra cómo crear una tabla con el rendimiento en caliente habilitado.

Java

SDK para Java 2.x

Cree una tabla de DynamoDB con una configuración de rendimiento en caliente mediante AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
```

```
import software.amazon.awssdk.services.dynamodb.model.AttributeDefinition;
import software.amazon.awssdk.services.dynamodb.model.CreateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.CreateTableResponse;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndex;
import software.amazon.awssdk.services.dynamodb.model.KeySchemaElement;
import software.amazon.awssdk.services.dynamodb.model.KeyType;
import software.amazon.awssdk.services.dynamodb.model.Projection;
import software.amazon.awssdk.services.dynamodb.model.ProvisionedThroughput;
import software.amazon.awssdk.services.dynamodb.model.ScalarAttributeType;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;

    public static WarmThroughput buildWarmThroughput(final Long
 readUnitsPerSecond, final Long writeUnitsPerSecond) {
        return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
            .build();
    }

    /**
     * Builds a ProvisionedThroughput object with the specified read and write
 capacity units.
     *
     * @param readCapacityUnits The read capacity units
     * @param writeCapacityUnits The write capacity units
     * @return A configured ProvisionedThroughput object
     */
    public static ProvisionedThroughput buildProvisionedThroughput(
        final Long readCapacityUnits, final Long writeCapacityUnits) {
        return ProvisionedThroughput.builder()
            .readCapacityUnits(readCapacityUnits)
            .writeCapacityUnits(writeCapacityUnits)
            .build();
    }

    /**
     * Builds an AttributeDefinition with the specified name and type.
     *
     * @param attributeName The attribute name
     * @param scalarAttributeType The attribute type
     * @return A configured AttributeDefinition
     */
    private static AttributeDefinition buildAttributeDefinition(
```

```
        final String attributeName, final ScalarAttributeType
scalarAttributeType) {
        return AttributeDefinition.builder()
            .attributeName(attributeName)
            .attributeType(scalarAttributeType)
            .build();
    }

    /**
     * Builds a KeySchemaElement with the specified name and key type.
     *
     * @param attributeName The attribute name
     * @param keyType The key type (HASH or RANGE)
     * @return A configured KeySchemaElement
     */
    private static KeySchemaElement buildKeySchemaElement(final String
attributeName, final KeyType keyType) {
        return KeySchemaElement.builder()
            .attributeName(attributeName)
            .keyType(keyType)
            .build();
    }

    /**
     * Creates a DynamoDB table with the specified configuration including warm
throughput settings.
     *
     * @param ddb The DynamoDB client
     * @param tableName The name of the table to create
     * @param partitionKey The partition key attribute name
     * @param sortKey The sort key attribute name
     * @param miscellaneousKeyAttribute Additional key attribute name for GSI
     * @param nonKeyAttribute Non-key attribute to include in GSI projection
     * @param tableReadCapacityUnits Read capacity units for the table
     * @param tableWriteCapacityUnits Write capacity units for the table
     * @param tableWarmReadUnitsPerSecond Warm read units per second for the
table
     * @param tableWarmWriteUnitsPerSecond Warm write units per second for the
table
     * @param globalSecondaryIndexName The name of the GSI to create
     * @param globalSecondaryIndexReadCapacityUnits Read capacity units for the
GSI
     * @param globalSecondaryIndexWriteCapacityUnits Write capacity units for the
GSI
```

```
    * @param globalSecondaryIndexWarmReadUnitsPerSecond Warm read units per
second for the GSI
    * @param globalSecondaryIndexWarmWriteUnitsPerSecond Warm write units per
second for the GSI
    */
   public static void createDynamoDBTable(
       final DynamoDbClient ddb,
       final String tableName,
       final String partitionKey,
       final String sortKey,
       final String miscellaneousKeyAttribute,
       final String nonKeyAttribute,
       final Long tableReadCapacityUnits,
       final Long tableWriteCapacityUnits,
       final Long tableWarmReadUnitsPerSecond,
       final Long tableWarmWriteUnitsPerSecond,
       final String globalSecondaryIndexName,
       final Long globalSecondaryIndexReadCapacityUnits,
       final Long globalSecondaryIndexWriteCapacityUnits,
       final Long globalSecondaryIndexWarmReadUnitsPerSecond,
       final Long globalSecondaryIndexWarmWriteUnitsPerSecond) {

       // Define the table attributes
       final AttributeDefinition partitionKeyAttribute =
buildAttributeDefinition(partitionKey, ScalarAttributeType.S);
       final AttributeDefinition sortKeyAttribute =
buildAttributeDefinition(sortKey, ScalarAttributeType.S);
       final AttributeDefinition miscellaneousKeyAttributeDefinition =
           buildAttributeDefinition(miscellaneousKeyAttribute,
ScalarAttributeType.N);
       final AttributeDefinition[] attributeDefinitions = {
           partitionKeyAttribute, sortKeyAttribute,
miscellaneousKeyAttributeDefinition
       };

       // Define the table key schema
       final KeySchemaElement partitionKeyElement =
buildKeySchemaElement(partitionKey, KeyType.HASH);
       final KeySchemaElement sortKeyElement = buildKeySchemaElement(sortKey,
KeyType.RANGE);
       final KeySchemaElement[] keySchema = {partitionKeyElement,
sortKeyElement};

       // Define the provisioned throughput for the table
```

```
        final ProvisionedThroughput provisionedThroughput =
            buildProvisionedThroughput(tableReadCapacityUnits,
tableWriteCapacityUnits);

        // Define the Global Secondary Index (GSI)
        final KeySchemaElement globalSecondaryIndexPartitionKeyElement =
buildKeySchemaElement(sortKey, KeyType.HASH);
        final KeySchemaElement globalSecondaryIndexSortKeyElement =
            buildKeySchemaElement(miscellaneousKeyAttribute, KeyType.RANGE);
        final KeySchemaElement[] gsiKeySchema = {
            globalSecondaryIndexPartitionKeyElement,
globalSecondaryIndexSortKeyElement
        };

        final Projection gsiProjection = Projection.builder()
            .projectionType(PROJECTION_TYPE_INCLUDE)
            .nonKeyAttributes(nonKeyAttribute)
            .build();

        final ProvisionedThroughput gsiProvisionedThroughput =
            buildProvisionedThroughput(globalSecondaryIndexReadCapacityUnits,
globalSecondaryIndexWriteCapacityUnits);

        // Define the warm throughput for the Global Secondary Index (GSI)
        final WarmThroughput gsiWarmThroughput = buildWarmThroughput(
            globalSecondaryIndexWarmReadUnitsPerSecond,
globalSecondaryIndexWarmWriteUnitsPerSecond);

        final GlobalSecondaryIndex globalSecondaryIndex =
GlobalSecondaryIndex.builder()
            .indexName(globalSecondaryIndexName)
            .keySchema(gsiKeySchema)
            .projection(gsiProjection)
            .provisionedThroughput(gsiProvisionedThroughput)
            .warmThroughput(gsiWarmThroughput)
            .build();

        // Define the warm throughput for the table
        final WarmThroughput tableWarmThroughput =
            buildWarmThroughput(tableWarmReadUnitsPerSecond,
tableWarmWriteUnitsPerSecond);

        final CreateTableRequest request = CreateTableRequest.builder()
            .tableName(tableName)
```

```
            .attributeDefinitions(attributeDefinitions)
            .keySchema(keySchema)
            .provisionedThroughput(provisionedThroughput)
            .globalSecondaryIndexes(globalSecondaryIndex)
            .warmThroughput(tableWarmThroughput)
            .build();

        final CreateTableResponse response = ddb.createTable(request);
        System.out.println(response);
    }
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Cree una tabla de DynamoDB con una configuración de rendimiento en caliente mediante AWS SDK para JavaScript.

```
import { DynamoDBClient, CreateTableCommand } from "@aws-sdk/client-dynamodb";

export async function createDynamoDBTableWithWarmThroughput(
  tableName,
  partitionKey,
  sortKey,
  miscKeyAttr,
  nonKeyAttr,
  tableProvisionedReadUnits,
  tableProvisionedWriteUnits,
  tableWarmReads,
  tableWarmWrites,
  indexName,
  indexProvisionedReadUnits,
  indexProvisionedWriteUnits,
  indexWarmReads,
  indexWarmWrites,
  region = "us-east-1"
) {
  try {
```

```
    const ddbClient = new DynamoDBClient({ region: region });
    const command = new CreateTableCommand({
      TableName: tableName,
      AttributeDefinitions: [
          { AttributeName: partitionKey, AttributeType: "S" },
          { AttributeName: sortKey, AttributeType: "S" },
          { AttributeName: miscKeyAttr, AttributeType: "N" },
      ],
      KeySchema: [
          { AttributeName: partitionKey, KeyType: "HASH" },
          { AttributeName: sortKey, KeyType: "RANGE" },
      ],
      ProvisionedThroughput: {
          ReadCapacityUnits: tableProvisionedReadUnits,
          WriteCapacityUnits: tableProvisionedWriteUnits,
      },
      WarmThroughput: {
          ReadUnitsPerSecond: tableWarmReads,
          WriteUnitsPerSecond: tableWarmWrites,
      },
      GlobalSecondaryIndexes: [
          {
            IndexName: indexName,
            KeySchema: [
                { AttributeName: sortKey, KeyType: "HASH" },
                { AttributeName: miscKeyAttr, KeyType: "RANGE" },
            ],
            Projection: {
                ProjectionType: "INCLUDE",
                NonKeyAttributes: [nonKeyAttr],
            },
            ProvisionedThroughput: {
                ReadCapacityUnits: indexProvisionedReadUnits,
                WriteCapacityUnits: indexProvisionedWriteUnits,
            },
            WarmThroughput: {
                ReadUnitsPerSecond: indexWarmReads,
                WriteUnitsPerSecond: indexWarmWrites,
            },
          },
      ],
    });
    const response = await ddbClient.send(command);
    console.log(response);
```

```
    return response;
  } catch (error) {
    console.error(`Error creating table: ${error}`);
    throw error;
  }
}

// Example usage (commented out for testing)
/*
createDynamoDBTableWithWarmThroughput(
  'example-table',
  'pk',
  'sk',
  'gsiKey',
  'data',
  10, 10, 5, 5,
  'example-index',
  5, 5, 2, 2
);
*/
```

- Para obtener información sobre la API, consulte CreateTable en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Cree una tabla de DynamoDB con una configuración de rendimiento en caliente mediante AWS SDK for Python (Boto3).

```
from boto3 import client
from botocore.exceptions import ClientError


def create_dynamodb_table_warm_throughput(
    table_name,
    partition_key,
    sort_key,
    misc_key_attr,
```

```
    non_key_attr,
    table_provisioned_read_units,
    table_provisioned_write_units,
    table_warm_reads,
    table_warm_writes,
    gsi_name,
    gsi_provisioned_read_units,
    gsi_provisioned_write_units,
    gsi_warm_reads,
    gsi_warm_writes,
    region_name="us-east-1",
):
    """
    Creates a DynamoDB table with a warm throughput setting configured.

    :param table_name: The name of the table to be created.
    :param partition_key: The partition key for the table being created.
    :param sort_key: The sort key for the table being created.
    :param misc_key_attr: A miscellaneous key attribute for the table being
 created.
    :param non_key_attr: A non-key attribute for the table being created.
    :param table_provisioned_read_units: The newly created table's provisioned
 read capacity units.
    :param table_provisioned_write_units: The newly created table's provisioned
 write capacity units.
    :param table_warm_reads: The read units per second setting for the table's
 warm throughput.
    :param table_warm_writes: The write units per second setting for the table's
 warm throughput.
    :param gsi_name: The name of the Global Secondary Index (GSI) to be created
 on the table.
    :param gsi_provisioned_read_units: The configured Global Secondary Index
(GSI) provisioned read capacity units.
    :param gsi_provisioned_write_units: The configured Global Secondary Index
(GSI) provisioned write capacity units.
    :param gsi_warm_reads: The read units per second setting for the Global
Secondary Index (GSI)'s warm throughput.
    :param gsi_warm_writes: The write units per second setting for the Global
Secondary Index (GSI)'s warm throughput.
    :param region_name: The AWS Region name to target. defaults to us-east-1
    """
    try:
        ddb = client("dynamodb", region_name=region_name)
```

```
        # Define the table attributes
        attribute_definitions = [
            {"AttributeName": partition_key, "AttributeType": "S"},
            {"AttributeName": sort_key, "AttributeType": "S"},
            {"AttributeName": misc_key_attr, "AttributeType": "N"},
        ]

        # Define the table key schema
        key_schema = [
            {"AttributeName": partition_key, "KeyType": "HASH"},
            {"AttributeName": sort_key, "KeyType": "RANGE"},
        ]

        # Define the provisioned throughput for the table
        provisioned_throughput = {
            "ReadCapacityUnits": table_provisioned_read_units,
            "WriteCapacityUnits": table_provisioned_write_units,
        }

        # Define the global secondary index
        gsi_key_schema = [
            {"AttributeName": sort_key, "KeyType": "HASH"},
            {"AttributeName": misc_key_attr, "KeyType": "RANGE"},
        ]
        gsi_projection = {"ProjectionType": "INCLUDE", "NonKeyAttributes":
[non_key_attr]}
        gsi_provisioned_throughput = {
            "ReadCapacityUnits": gsi_provisioned_read_units,
            "WriteCapacityUnits": gsi_provisioned_write_units,
        }
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_warm_reads,
            "WriteUnitsPerSecond": gsi_warm_writes,
        }
        global_secondary_indexes = [
            {
                "IndexName": gsi_name,
                "KeySchema": gsi_key_schema,
                "Projection": gsi_projection,
                "ProvisionedThroughput": gsi_provisioned_throughput,
                "WarmThroughput": gsi_warm_throughput,
            }
        ]
```

```
        # Define the warm throughput for the table
        warm_throughput = {
            "ReadUnitsPerSecond": table_warm_reads,
            "WriteUnitsPerSecond": table_warm_writes,
        }

        # Create the DynamoDB client and create the table
        response = ddb.create_table(
            TableName=table_name,
            AttributeDefinitions=attribute_definitions,
            KeySchema=key_schema,
            ProvisionedThroughput=provisioned_throughput,
            GlobalSecondaryIndexes=global_secondary_indexes,
            WarmThroughput=warm_throughput,
        )

        print(response)
        return response
    except ClientError as e:
        print(f"Error creating table: {e}")
        raise e
```

- Para obtener detalles de la API, consulte CreateTable en la Referencia de la API del SDK de AWS para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Creación de una aplicación web para hacer un seguimiento de los datos de DynamoDB

Los siguientes ejemplos de código muestran cómo crear una aplicación web que realice un seguimiento de los elementos de trabajo de una tabla de Amazon DynamoDB y use Amazon Simple Email Service (Amazon SES) para enviar informes.

.NET

SDK para .NET

Muestra cómo utilizar la API de .NET de Amazon DynamoDB para crear una aplicación web dinámica que haga un seguimiento de los datos de trabajo de DynamoDB.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- DynamoDB
- Amazon SES

Java

SDK para Java 2.x

Muestra cómo utilizar la API de Amazon DynamoDB para crear una aplicación web dinámica que haga un seguimiento de los datos de trabajo de DynamoDB.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- DynamoDB
- Amazon SES

Kotlin

SDK para Kotlin

Muestra cómo utilizar la API de Amazon DynamoDB para crear una aplicación web dinámica que haga un seguimiento de los datos de trabajo de DynamoDB.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- DynamoDB

- Amazon SES

Python

SDK para Python (Boto3)

Muestra cómo utilizar AWS SDK for Python (Boto3) para crear un servicio REST que haga un seguimiento de los elementos de trabajo de Amazon DynamoDB y envíe informes por correo electrónico mediante Amazon Simple Email Service (Amazon SES). En este ejemplo se utiliza el marco web de Flask para gestionar el enrutamiento HTTP y se integra con una página web de React para presentar una aplicación web completamente funcional.

- Cree un servicio REST de Flask que se integre con Servicios de AWS.
- Lea, escriba y actualice los elementos de trabajo almacenados en una tabla de DynamoDB.
- Utilice Amazon SES para enviar informes de elementos de trabajo por correo electrónico.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en el Repositorio de ejemplos de código de AWS en GitHub.

Servicios utilizados en este ejemplo

- DynamoDB
- Amazon SES

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Creación una aplicación de chat de websocket con API Gateway

En el siguiente ejemplo se muestra cómo crear una aplicación de chat servida por una API de websocket basada en Amazon API Gateway.

Python

SDK para Python (Boto3)

Muestra cómo utilizar AWS SDK for Python (Boto3) con Amazon API Gateway V2 para crear una API de websocket que se integre con AWS Lambda y Amazon DynamoDB.

- Crear una API de websocket servida por API Gateway.

- Definir un identificador Lambda que almacene las conexiones en DynamoDB y envíe mensajes a otros participantes del chat.
- Conectar con la aplicación de chat de websocket y enviar mensajes con el paquete Websockets.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Creación de un elemento de DynamoDB con un TTL mediante un AWS SDK

Los siguientes ejemplos de código muestran cómo crear un elemento con TTL.

Java

SDK para Java 2.x

```
package com.amazon.samplelib.ttl;

import com.amazon.samplelib.CodeSampleUtils;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutItemRequest;
import software.amazon.awssdk.services.dynamodb.model.PutItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
```

```
import java.util.Optional;

/**
 * Creates an item in a DynamoDB table with TTL attributes.
 * This class demonstrates how to add TTL expiration timestamps to DynamoDB
 items.
 */
public class CreateTTL {

    private static final String USAGE =
        """
            Usage:
                <tableName> <primaryKey> <sortKey> <region>
            Where:
                tableName - The Amazon DynamoDB table being queried.
                primaryKey - The name of the primary key. Also known as the hash
 or partition key.
                sortKey - The name of the sort key. Also known as the range
 attribute.
                region (optional) - The AWS region that the Amazon DynamoDB table
 is located in. (Default: us-east-1)
            """;
    private static final int DAYS_TO_EXPIRE = 90;
    private static final int SECONDS_PER_DAY = 24 * 60 * 60;
    private static final String PRIMARY_KEY_ATTR = "primaryKey";
    private static final String SORT_KEY_ATTR = "sortKey";
    private static final String CREATION_DATE_ATTR = "creationDate";
    private static final String EXPIRE_AT_ATTR = "expireAt";
    private static final String SUCCESS_MESSAGE = "%s PutItem operation with TTL
 successful.";
    private static final String TABLE_NOT_FOUND_ERROR = "Error: The Amazon
 DynamoDB table \"%s\" can't be found.";

    private final DynamoDbClient dynamoDbClient;

    /**
     * Constructs a CreateTTL instance with the specified DynamoDB client.
     *
     * @param dynamoDbClient The DynamoDB client to use
     */
    public CreateTTL(final DynamoDbClient dynamoDbClient) {
        this.dynamoDbClient = dynamoDbClient;
    }
```

```
    /**
     * Constructs a CreateTTL with a default DynamoDB client.
     */
    public CreateTTL() {
        this.dynamoDbClient = null;
    }

    /**
     * Main method to demonstrate creating an item with TTL.
     *
     * @param args Command line arguments
     */
    public static void main(final String[] args) {
        try {
            int result = new CreateTTL().processArgs(args);
            System.exit(result);
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Process command line arguments and create an item with TTL.
     *
     * @param args Command line arguments
     * @return 0 if successful, non-zero otherwise
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     * @throws IllegalArgumentException If arguments are invalid
     */
    public int processArgs(final String[] args) {
        // Argument validation (remove or replace this line when reusing this
code)
        CodeSampleUtils.validateArgs(args, new int[] {3, 4}, USAGE);

        final String tableName = args[0];
        final String primaryKey = args[1];
        final String sortKey = args[2];
        final Region region = Optional.ofNullable(args.length > 3 ? args[3] :
null)
            .map(Region::of)
            .orElse(Region.US_EAST_1);
```

```
        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final CreateTTL createTTL = new CreateTTL(ddb);
            createTTL.createItemWithTTL(tableName, primaryKey, sortKey);
            return 0;
        } catch (Exception e) {
            throw e;
        }
    }

    /**
     * Creates an item in the specified table with TTL attributes.
     *
     * @param tableName The name of the table
     * @param primaryKeyValue The value for the primary key
     * @param sortKeyValue The value for the sort key
     * @return The response from the PutItem operation
     * @throws ResourceNotFoundException If the table doesn't exist
     * @throws DynamoDbException If an error occurs during the operation
     */
    public PutItemResponse createItemWithTTL(
        final String tableName, final String primaryKeyValue, final String
sortKeyValue) {
        // Get current time in epoch second format
        final long createDate = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = createDate + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        final Map<String, AttributeValue> itemMap = new HashMap<>();
        itemMap.put(
            PRIMARY_KEY_ATTR,
AttributeValue.builder().s(primaryKeyValue).build());
        itemMap.put(SORT_KEY_ATTR,
AttributeValue.builder().s(sortKeyValue).build());
        itemMap.put(
            CREATION_DATE_ATTR,
            AttributeValue.builder().n(String.valueOf(createDate)).build());
        itemMap.put(
            EXPIRE_AT_ATTR,
            AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final PutItemRequest request =
```

```
            PutItemRequest.builder().tableName(tableName).item(itemMap).build();

        try {
            final PutItemResponse response = dynamoDbClient.putItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

```
import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";

export function createDynamoDBItem(table_name, region, partition_key, sort_key) {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    // Get the current time in epoch second format
    const current_time = Math.floor(new Date().getTime() / 1000);

    // Calculate the expireAt time (90 days from now) in epoch second format
    const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 *
 1000) / 1000);

    // Create DynamoDB item
    const item = {
        'partitionKey': {'S': partition_key},
```

```
        'sortKey': {'S': sort_key},
        'createdAt': {'N': current_time.toString()},
        'expireAt': {'N': expire_at.toString()}
    };

    const putItemCommand = new PutItemCommand({
        TableName: table_name,
        Item: item,
        ProvisionedThroughput: {
            ReadCapacityUnits: 1,
            WriteCapacityUnits: 1,
        },
    });

    client.send(putItemCommand, function(err, data) {
        if (err) {
            console.log("Exception encountered when creating item %s, here's what
 happened: ", data, err);
            throw err;
        } else {
            console.log("Item created successfully: %s.", data);
            return data;
        }
    });
}

// Example usage (commented out for testing)
// createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
 'your-sort-key-value');
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

```
from datetime import datetime, timedelta

import boto3
```

```
def create_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Creates a DynamoDB item with an attached expiry attribute.

    :param table_name: Table name for the boto3 resource to target when creating
 an item
    :param region: string representing the AWS region. Example: `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expiration time (90 days from now) in epoch second format
        expiration_time = int((datetime.now() + timedelta(days=90)).timestamp())

        item = {
            "primaryKey": primary_key,
            "sortKey": sort_key,
            "creationDate": current_time,
            "expireAt": expiration_time,
        }
        response = table.put_item(Item=item)

        print("Item created successfully.")
        return response
    except Exception as e:
        print(f"Error creating item: {e}")
        raise e


# Use your own values
create_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
)
```

- Para obtener información sobre la API, consulte PutItem en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Creación y administración de tablas globales de DynamoDB con coherencia alta de varias regiones mediante un AWS SDK

Los ejemplos de código siguientes, muestran cómo crear y administrar las tablas globales de DynamoDB con coherencia alta de varias regiones (MRSC).

- Cree una tabla con coherencia alta de varias regiones.
- Verifique la configuración de MRSC y el estado de la réplica.
- Pruebe la coherencia alta entre regiones con lecturas inmediatas.
- Realice escrituras condicionales con garantías de MRSC.
- Limpie los recursos de la tabla global de MRSC.

Bash

AWS CLI con Bash script

Cree una tabla con coherencia alta de varias regiones.

```
# Step 1: Create a new table in us-east-2 (primary region for MRSC)
# Note: Table must be empty when enabling MRSC
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --region us-east-2

# Wait for table to become active
```

```
aws dynamodb wait table-exists --table-name MusicTable --region us-east-2

# Step 2: Add replica and witness with Multi-Region Strong Consistency
# MRSC requires exactly three replicas in supported regions
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Create": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Create": {"RegionName": "us-west-2"}}]' \
    --multi-region-consistency STRONG \
    --region us-east-2
```

Verifique la configuración de MRSC y el estado de la réplica.

```
# Verify the global table configuration and MRSC setting
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

Pruebe la coherencia alta con lecturas inmediatas entre regiones.

```
# Write an item to the primary region
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"},"Album":
 {"S":"The Beatles 1967-1970"},"Year": {"N":"1968"}}' \
    --region us-east-2

# Read the item from replica region to verify strong consistency (cannot read or
 write to witness)
# No wait time needed - MRSC provides immediate consistency
echo "Reading from us-east-1 (immediate consistency):"
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --consistent-read \
    --region us-east-1
```

Realice escrituras condicionales con garantías de MRSC.

```
# Perform a conditional update from a different region
# This demonstrates that conditions work consistently across all regions
aws dynamodb update-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"The Beatles"},"SongTitle": {"S":"Hey Jude"}}' \
    --update-expression "SET #rating = :rating" \
    --condition-expression "attribute_exists(Artist)" \
    --expression-attribute-names '{"#rating": "Rating"}' \
    --expression-attribute-values '{":rating": {"N":"5"}}' \
    --region us-east-1
```

Limpie los recursos de la tabla global de MRSC.

```
# Remove replica tables (must be done before deleting the primary table)
aws dynamodb update-table \
    --table-name MusicTable \
    --replica-updates '[{"Delete": {"RegionName": "us-east-1"}}]' \
    --global-table-witness-updates '[{"Delete": {"RegionName": "us-west-2"}}]' \
    --region us-east-2

# Wait for replicas to be deleted
echo "Waiting for replicas to be deleted..."
sleep 30

# Delete the primary table
aws dynamodb delete-table \
    --table-name MusicTable \
    --region us-east-2
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de comandos de AWS CLI.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem

- UpdateItem
- UpdateTable

Java

SDK para Java 2.x

Cree una tabla regional lista para la conversión de MRSC con AWS SDK for Java 2.x.

```
   public static CreateTableResponse createRegionalTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Creating regional table: " + tableName + " (must be
empty for MRSC)");

           CreateTableRequest createTableRequest = CreateTableRequest.builder()
               .tableName(tableName)
               .attributeDefinitions(
                   AttributeDefinition.builder()
                       .attributeName("Artist")
                       .attributeType(ScalarAttributeType.S)
                       .build(),
                   AttributeDefinition.builder()
                       .attributeName("SongTitle")
                       .attributeType(ScalarAttributeType.S)
                       .build())
               .keySchema(
                   KeySchemaElement.builder()
                       .attributeName("Artist")
                       .keyType(KeyType.HASH)
                       .build(),
                   KeySchemaElement.builder()
                       .attributeName("SongTitle")
```

```
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Regional table creation initiated. Status: "
                + response.tableDescription().tableStatus());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create regional table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to create regional table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Convierta una tabla regional en MRSC utilizando réplicas y testigos con AWS SDK for Java 2.x.

```
    public static UpdateTableResponse convertToMRSCWithWitness(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
```

```
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("Converting table to MRSC with replica in " +
replicaRegion.id() + " and witness in "
                + witnessRegion.id());

            // Create replica update using ReplicationGroupUpdate
            ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
                .create(CreateReplicationGroupMemberAction.builder()
                    .regionName(replicaRegion.id())
                    .build())
                .build();

            // Create witness update
            GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
                .create(CreateGlobalTableWitnessGroupMemberAction.builder()
                    .regionName(witnessRegion.id())
                    .build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .multiRegionConsistency(MultiRegionConsistency.STRONG)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC conversion initiated. Status: "
                + response.tableDescription().tableStatus());
            LOGGER.info("UpdateTableResponse full object: " + response);
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to convert table to MRSC: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to convert table to MRSC: " + tableName)
                .cause(e)
```

```
                .build();
        }
    }
```

Describa la configuración de una tabla global de MRSC con AWS SDK for Java 2.x.

```
   public static DescribeTableResponse describeMRSCTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Describing MRSC global table: " + tableName);

           DescribeTableRequest request =
               DescribeTableRequest.builder().tableName(tableName).build();

           DescribeTableResponse response =
dynamoDbClient.describeTable(request);

           LOGGER.info("Table status: " + response.table().tableStatus());
           LOGGER.info("Multi-region consistency: " +
response.table().multiRegionConsistency());

           if (response.table().replicas() != null
               && !response.table().replicas().isEmpty()) {
               LOGGER.info("Number of replicas: " +
response.table().replicas().size());
               response.table()
                   .replicas()
                   .forEach(replica -> LOGGER.info(
                       "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
           }

           if (response.table().globalTableWitnesses() != null
```

```
                && !response.table().globalTableWitnesses().isEmpty()) {
                LOGGER.info("Number of witnesses: "
                    + response.table().globalTableWitnesses().size());
                response.table()
                    .globalTableWitnesses()
                    .forEach(witness -> LOGGER.info(
                        "Witness region: " + witness.regionName() + ", Status: "
+ witness.witnessStatus()));
            }

            return response;

        } catch (ResourceNotFoundException e) {
            LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Table not found: " + tableName)
                .cause(e)
                .build();
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to describe table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Agregue elementos de prueba para verificar la coherencia alta de MRSC con AWS SDK for Java 2.x.

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String album,
        final String year) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
```

```
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Adding test item to MRSC global table: " + tableName);

            Map<String, AttributeValue> item = new HashMap<>();
            item.put("Artist", AttributeValue.builder().s(artist).build());
            item.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            if (album != null && !album.trim().isEmpty()) {
                item.put("Album", AttributeValue.builder().s(album).build());
            }
            if (year != null && !year.trim().isEmpty()) {
                item.put("Year", AttributeValue.builder().n(year).build());
            }

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully with strong consistency");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to add test item to table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Lea los elementos con lecturas coherentes de las réplicas de MRSC con AWS SDK for Java 2.x.

```
   public static GetItemResponse getItemWithConsistentRead(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (artist == null || artist.trim().isEmpty()) {
           throw new IllegalArgumentException("Artist cannot be null or empty");
       }
       if (songTitle == null || songTitle.trim().isEmpty()) {
           throw new IllegalArgumentException("Song title cannot be null or
empty");
       }

       try {
           LOGGER.info("Reading item from MRSC global table with consistent
read: " + tableName);

           Map<String, AttributeValue> key = new HashMap<>();
           key.put("Artist", AttributeValue.builder().s(artist).build());
           key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

           GetItemRequest getItemRequest = GetItemRequest.builder()
               .tableName(tableName)
               .key(key)
               .consistentRead(true)
               .build();

           GetItemResponse response = dynamoDbClient.getItem(getItemRequest);

           if (response.hasItem()) {
               LOGGER.info("Item found with strong consistency - no wait time
needed");
```

```
            } else {
                LOGGER.info("Item not found");
            }

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to read item from table: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to read item from table: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Realice actualizaciones condicionales con garantías de MRSC mediante AWS SDK for Java 2.x.

```
    public static UpdateItemResponse performConditionalUpdate(
        final DynamoDbClient dynamoDbClient,
        final String tableName,
        final String artist,
        final String songTitle,
        final String rating) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }
        if (rating == null || rating.trim().isEmpty()) {
            throw new IllegalArgumentException("Rating cannot be null or empty");
```

```
        }

        try {
            LOGGER.info("Performing conditional update on MRSC global table: " +
tableName);

            Map<String, AttributeValue> key = new HashMap<>();
            key.put("Artist", AttributeValue.builder().s(artist).build());
            key.put("SongTitle", AttributeValue.builder().s(songTitle).build());

            Map<String, String> expressionAttributeNames = new HashMap<>();
            expressionAttributeNames.put("#rating", "Rating");

            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
            expressionAttributeValues.put(
                ":rating", AttributeValue.builder().n(rating).build());

            UpdateItemRequest updateItemRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #rating = :rating")
                .conditionExpression("attribute_exists(Artist)")
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .build();

            UpdateItemResponse response =
dynamoDbClient.updateItem(updateItemRequest);
            LOGGER.info("Conditional update successful - demonstrates strong
consistency");

            return response;

        } catch (ConditionalCheckFailedException e) {
            LOGGER.warning("Conditional check failed: " + e.getMessage());
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to perform conditional update: " + tableName +
" - " + e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to perform conditional update: " + tableName)
                .cause(e)
                .build();
```

```
        }
    }
```

Espere a que las réplicas de MRSC y los testigos se activen mediante AWS SDK for Java 2.x.

```
    public static void waitForMRSCReplicasActive(
        final DynamoDbClient dynamoDbClient, final String tableName, final int
maxWaitTimeSeconds)
        throws InterruptedException {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (maxWaitTimeSeconds <= 0) {
            throw new IllegalArgumentException("Max wait time must be positive");
        }

        try {
            LOGGER.info("Waiting for MRSC replicas and witnesses to become
active: " + tableName);

            final long startTime = System.currentTimeMillis();
            final long maxWaitTimeMillis = maxWaitTimeSeconds * 1000L;
            int backoffSeconds = 5; // Start with 5 second intervals
            final int maxBackoffSeconds = 30; // Cap at 30 seconds

            while (System.currentTimeMillis() - startTime < maxWaitTimeMillis) {
                DescribeTableResponse response =
describeMRSCTable(dynamoDbClient, tableName);

                boolean allActive = true;
                StringBuilder statusReport = new StringBuilder();

                if (response.table().multiRegionConsistency() == null
                    || !MultiRegionConsistency.STRONG
                        .toString()

.equals(response.table().multiRegionConsistency().toString())) {
```

```
                    allActive = false;
                    statusReport
                        .append("MultiRegionConsistency: ")
                        .append(response.table().multiRegionConsistency())
                        .append(" ");
                }
                if (response.table().replicas() == null
                    || response.table().replicas().isEmpty()) {
                    allActive = false;
                    statusReport.append("No replicas found. ");
                }
                if (response.table().globalTableWitnesses() == null
                    || response.table().globalTableWitnesses().isEmpty()) {
                    allActive = false;
                    statusReport.append("No witnesses found. ");
                }

                // Check table status
                if (!"ACTIVE".equals(response.table().tableStatus().toString()))
{
                    allActive = false;
                    statusReport
                        .append("Table: ")
                        .append(response.table().tableStatus())
                        .append(" ");
                }

                // Check replica status
                if (response.table().replicas() != null) {
                    for (var replica : response.table().replicas()) {
                        if (!"ACTIVE".equals(replica.replicaStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Replica(")
                                .append(replica.regionName())
                                .append("): ")
                                .append(replica.replicaStatus())
                                .append(" ");
                        }
                    }
                }

                // Check witness status
```

```
                if (response.table().globalTableWitnesses() != null) {
                    for (var witness : response.table().globalTableWitnesses()) {
                        if (!"ACTIVE".equals(witness.witnessStatus().toString()))
{
                            allActive = false;
                            statusReport
                                .append("Witness(")
                                .append(witness.regionName())
                                .append("): ")
                                .append(witness.witnessStatus())
                                .append(" ");
                        }
                    }
                }

                if (allActive) {
                    LOGGER.info("All MRSC replicas and witnesses are now active:
" + tableName);
                    return;
                }

                LOGGER.info("Waiting for MRSC components to become active.
Status: " + statusReport.toString());
                LOGGER.info("Next check in " + backoffSeconds + " seconds...");

                tempWait(backoffSeconds);

                // Exponential backoff with cap
                backoffSeconds = Math.min(backoffSeconds * 2, maxBackoffSeconds);
            }

            throw DynamoDbException.builder()
                .message("Timeout waiting for MRSC replicas to become active
after " + maxWaitTimeSeconds + " seconds")
                .build();

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("Failed to wait for MRSC replicas to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

Limpie las réplicas y los testigos de MRSC mediante AWS SDK for Java 2.x.

```
   public static UpdateTableResponse cleanupMRSCReplicas(
       final DynamoDbClient dynamoDbClient,
       final String tableName,
       final Region replicaRegion,
       final Region witnessRegion) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (replicaRegion == null) {
           throw new IllegalArgumentException("Replica region cannot be null");
       }
       if (witnessRegion == null) {
           throw new IllegalArgumentException("Witness region cannot be null");
       }

       try {
           LOGGER.info("Cleaning up MRSC replicas and witnesses for table: " +
tableName);

           // Remove replica using ReplicationGroupUpdate
           ReplicationGroupUpdate replicaUpdate =
ReplicationGroupUpdate.builder()
               .delete(DeleteReplicationGroupMemberAction.builder()
                   .regionName(replicaRegion.id())
                   .build())
               .build();

           // Remove witness
           GlobalTableWitnessGroupUpdate witnessUpdate =
GlobalTableWitnessGroupUpdate.builder()
               .delete(DeleteGlobalTableWitnessGroupMemberAction.builder()
                   .regionName(witnessRegion.id())
                   .build())
               .build();

           UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
               .tableName(tableName)
```

```
                .replicaUpdates(List.of(replicaUpdate))
                .globalTableWitnessUpdates(List.of(witnessUpdate))
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("MRSC cleanup initiated - removing replica and witness.
Response: " + response);

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to cleanup MRSC replicas: " + tableName + " - "
+ e.getMessage());
            throw DynamoDbException.builder()
                .message("Failed to cleanup MRSC replicas: " + tableName)
                .cause(e)
                .build();
        }
    }
```

Complete la demostración del flujo de trabajo de MRSC mediante AWS SDK for Java 2.x.

```
    public static void demonstrateCompleteMRSCWorkflow(
        final DynamoDbClient primaryClient,
        final DynamoDbClient replicaClient,
        final String tableName,
        final Region replicaRegion,
        final Region witnessRegion)
        throws InterruptedException {

        if (primaryClient == null) {
            throw new IllegalArgumentException("Primary DynamoDB client cannot be
null");
        }
        if (replicaClient == null) {
            throw new IllegalArgumentException("Replica DynamoDB client cannot be
null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
```

```
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }
        if (witnessRegion == null) {
            throw new IllegalArgumentException("Witness region cannot be null");
        }

        try {
            LOGGER.info("=== Starting Complete MRSC Workflow Demonstration ===");

            // Step 1: Create an empty single-Region table
            LOGGER.info("Step 1: Creating empty single-Region table");
            createRegionalTable(primaryClient, tableName);

            // Use the existing GlobalTableOperations method for basic table
waiting
            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 2: Convert to MRSC with replica and witness
            LOGGER.info("Step 2: Converting to MRSC with replica and witness");
            convertToMRSCWithWitness(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for MRSC conversion to complete using MRSC-specific waiter
            LOGGER.info("Waiting for MRSC conversion to complete...");
            waitForMRSCReplicasActive(primaryClient, tableName);

            LOGGER.info("Intermediate step: Waiting for table [" + tableName + "]
to become active before continuing");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // Step 3: Verify MRSC configuration
            LOGGER.info("Step 3: Verifying MRSC configuration");
            describeMRSCTable(primaryClient, tableName);

            // Step 4: Test strong consistency with data operations
            LOGGER.info("Step 4: Testing strong consistency with data
operations");

            // Add test item to primary region
```

```
            putTestItem(primaryClient, tableName, "The Beatles", "Hey Jude", "The
Beatles 1967-1970", "1968");

            // Immediately read from replica region (no wait needed with MRSC)
            LOGGER.info("Reading from replica region immediately (strong
consistency):");
            GetItemResponse getResponse =
                getItemWithConsistentRead(replicaClient, tableName, "The
Beatles", "Hey Jude");

            if (getResponse.hasItem()) {
                LOGGER.info("# Strong consistency verified - item immediately
available in replica region");
            } else {
                LOGGER.warning("# Item not found in replica region");
            }

            // Test conditional update from replica region
            LOGGER.info("Testing conditional update from replica region:");
            performConditionalUpdate(replicaClient, tableName, "The Beatles",
"Hey Jude", "5");
            LOGGER.info("# Conditional update successful - demonstrates strong
consistency");

            // Step 5: Cleanup
            LOGGER.info("Step 5: Cleaning up resources");
            cleanupMRSCReplicas(primaryClient, tableName, replicaRegion,
witnessRegion);

            // Wait for cleanup to complete using basic table waiter
            LOGGER.info("Waiting for replica cleanup to complete...");
            GlobalTableOperations.waitForTableActive(primaryClient, tableName);

            // "Halt" until replica/witness cleanup is complete
            DescribeTableResponse cleanupVerification =
describeMRSCTable(primaryClient, tableName);
            int backoffSeconds = 5; // Start with 5 second intervals
            while (cleanupVerification.table().multiRegionConsistency() != null)
{
                LOGGER.info("Waiting additional time (" + backoffSeconds + "
seconds) for MRSC cleanup to complete...");
                tempWait(backoffSeconds);

                // Exponential backoff with cap
```

```
                backoffSeconds = Math.min(backoffSeconds * 2, 30);
                cleanupVerification = describeMRSCTable(primaryClient,
tableName);
            }

            // Delete the primary table
            deleteTable(primaryClient, tableName);

            LOGGER.info("=== MRSC Workflow Demonstration Complete ===");
            LOGGER.info("");
            LOGGER.info("Key benefits of Multi-Region Strong Consistency
(MRSC):");
            LOGGER.info("- Immediate consistency across all regions (no eventual
consistency delays)");
            LOGGER.info("- Simplified application logic (no need to handle
eventual consistency)");
            LOGGER.info("- Support for conditional writes and transactions across
regions");
            LOGGER.info("- Consistent read operations from any region without
waiting");

        } catch (DynamoDbException | InterruptedException e) {
            LOGGER.severe("MRSC workflow failed: " + e.getMessage());
            throw e;
        }
    }
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for Java 2.x.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateItem
  - UpdateTable

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Creación y administración de tablas globales de DynamoDB que muestren MREC con un AWS SDK

En el ejemplo de código siguiente, se muestra cómo crear y administrar las tablas globales de DynamoDB con réplicas en varias regiones.

- Cree una tabla con un índice secundario global y flujos de DynamoDB.
- Agregue réplicas en diferentes regiones para crear una tabla global.
- Elimine réplicas de una tabla global.
- Agregue elementos de prueba para verificar la replicación en todas las regiones.
- Describa la configuración de la tabla global y el estado de las réplicas.

Java

SDK para Java 2.x

Cree una tabla con un índice secundario global y flujos de DynamoDB mediante AWS SDK for Java 2.x.

```
   public static CreateTableResponse createTableWithGSI(
       final DynamoDbClient dynamoDbClient, final String tableName, final String
indexName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (indexName == null || indexName.trim().isEmpty()) {
           throw new IllegalArgumentException("Index name cannot be null or
empty");
       }

       try {
```

```
            LOGGER.info("Creating table: " + tableName + " with GSI: " +
indexName);

            CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .tableName(tableName)
                .attributeDefinitions(
                    AttributeDefinition.builder()
                        .attributeName("Artist")
                        .attributeType(ScalarAttributeType.S)
                        .build(),
                    AttributeDefinition.builder()
                        .attributeName("SongTitle")
                        .attributeType(ScalarAttributeType.S)
                        .build())
                .keySchema(
                    KeySchemaElement.builder()
                        .attributeName("Artist")
                        .keyType(KeyType.HASH)
                        .build(),
                    KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.RANGE)
                        .build())
                .billingMode(BillingMode.PAY_PER_REQUEST)
                .globalSecondaryIndexes(GlobalSecondaryIndex.builder()
                    .indexName(indexName)
                    .keySchema(KeySchemaElement.builder()
                        .attributeName("SongTitle")
                        .keyType(KeyType.HASH)
                        .build())
                    .projection(

Projection.builder().projectionType(ProjectionType.ALL).build())
                    .build())
                .streamSpecification(StreamSpecification.builder()
                    .streamEnabled(true)
                    .streamViewType(StreamViewType.NEW_AND_OLD_IMAGES)
                    .build())
                .build();

            CreateTableResponse response =
dynamoDbClient.createTable(createTableRequest);
            LOGGER.info("Table creation initiated. Status: "
                + response.tableDescription().tableStatus());
```

```
            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to create table: " + tableName + " - " +
e.getMessage());
            throw e;
        }
    }
```

Espere a que una tabla se active mediante AWS SDK for Java 2.x.

```
    public static void waitForTableActive(final DynamoDbClient dynamoDbClient,
final String tableName) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }

        try {
            LOGGER.info("Waiting for table to become active: " + tableName);

            try (DynamoDbWaiter waiter =
                DynamoDbWaiter.builder().client(dynamoDbClient).build()) {
                DescribeTableRequest request =
                    DescribeTableRequest.builder().tableName(tableName).build();

                waiter.waitUntilTableExists(request);
                LOGGER.info("Table is now active: " + tableName);
            }

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to wait for table to become active: " +
tableName + " - " + e.getMessage());
            throw e;
        }
    }
```

Agregue una réplica para crear o ampliar una tabla global mediante AWS SDK for Java 2.x.

```
   public static UpdateTableResponse addReplica(
       final DynamoDbClient dynamoDbClient,
       final String tableName,
       final Region replicaRegion,
       final String indexName,
       final Long readCapacity) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }
       if (replicaRegion == null) {
           throw new IllegalArgumentException("Replica region cannot be null");
       }
       if (indexName == null || indexName.trim().isEmpty()) {
           throw new IllegalArgumentException("Index name cannot be null or
empty");
       }
       if (readCapacity == null || readCapacity <= 0) {
           throw new IllegalArgumentException("Read capacity must be a positive
number");
       }

       try {
           LOGGER.info("Adding replica in region: " + replicaRegion.id() + " for
table: " + tableName);

           // Create a ReplicationGroupUpdate for adding a replica
           ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
               .create(builder -> builder.regionName(replicaRegion.id())
                   .globalSecondaryIndexes(ReplicaGlobalSecondaryIndex.builder()
                       .indexName(indexName)

.provisionedThroughputOverride(ProvisionedThroughputOverride.builder()
                           .readCapacityUnits(readCapacity)
                           .build())
                       .build())
                   .build())
```

```
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica addition initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

Elimine una réplica de una tabla global mediante AWS SDK for Java 2.x.

```
    public static UpdateTableResponse removeReplica(
        final DynamoDbClient dynamoDbClient, final String tableName, final Region
replicaRegion) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (replicaRegion == null) {
            throw new IllegalArgumentException("Replica region cannot be null");
        }

        try {
            LOGGER.info("Removing replica in region: " + replicaRegion.id() + "
for table: " + tableName);
```

```
            // Create a ReplicationGroupUpdate for removing a replica
            ReplicationGroupUpdate replicationGroupUpdate =
ReplicationGroupUpdate.builder()
                .delete(builder ->
builder.regionName(replicaRegion.id()).build())
                .build();

            UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()
                .tableName(tableName)
                .replicaUpdates(replicationGroupUpdate)
                .build();

            UpdateTableResponse response =
dynamoDbClient.updateTable(updateTableRequest);
            LOGGER.info("Replica removal initiated in region: " +
replicaRegion.id());

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to remove replica in region: " +
replicaRegion.id() + " - " + e.getMessage());
            throw e;
        }
    }
```

Agregue elementos de prueba para verificar la replicación mediante AWS SDK for Java 2.x.

```
    public static PutItemResponse putTestItem(
        final DynamoDbClient dynamoDbClient, final String tableName, final String
artist, final String songTitle) {

        if (dynamoDbClient == null) {
            throw new IllegalArgumentException("DynamoDB client cannot be null");
        }
        if (tableName == null || tableName.trim().isEmpty()) {
            throw new IllegalArgumentException("Table name cannot be null or
empty");
        }
        if (artist == null || artist.trim().isEmpty()) {
            throw new IllegalArgumentException("Artist cannot be null or empty");
        }
```

```
        if (songTitle == null || songTitle.trim().isEmpty()) {
            throw new IllegalArgumentException("Song title cannot be null or
empty");
        }

        try {
            LOGGER.info("Adding test item to table: " + tableName);

            Map<String,
software.amazon.awssdk.services.dynamodb.model.AttributeValue> item = new
HashMap<>();
            item.put(
                "Artist",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(artist)
                    .build());
            item.put(
                "SongTitle",

software.amazon.awssdk.services.dynamodb.model.AttributeValue.builder()
                    .s(songTitle)
                    .build());

            PutItemRequest putItemRequest =
                PutItemRequest.builder().tableName(tableName).item(item).build();

            PutItemResponse response = dynamoDbClient.putItem(putItemRequest);
            LOGGER.info("Test item added successfully");

            return response;

        } catch (DynamoDbException e) {
            LOGGER.severe("Failed to add test item to table: " + tableName + " -
" + e.getMessage());
            throw e;
        }
    }
```

Describa la configuración de la tabla global y las réplicas mediante AWS SDK for Java 2.x.

```
   public static DescribeTableResponse describeTable(final DynamoDbClient
dynamoDbClient, final String tableName) {

       if (dynamoDbClient == null) {
           throw new IllegalArgumentException("DynamoDB client cannot be null");
       }
       if (tableName == null || tableName.trim().isEmpty()) {
           throw new IllegalArgumentException("Table name cannot be null or
empty");
       }

       try {
           LOGGER.info("Describing table: " + tableName);

           DescribeTableRequest request =
               DescribeTableRequest.builder().tableName(tableName).build();

           DescribeTableResponse response =
dynamoDbClient.describeTable(request);

           LOGGER.info("Table status: " + response.table().tableStatus());
           if (response.table().replicas() != null
               && !response.table().replicas().isEmpty()) {
               LOGGER.info("Number of replicas: " +
response.table().replicas().size());
               response.table()
                   .replicas()
                   .forEach(replica -> LOGGER.info(
                       "Replica region: " + replica.regionName() + ", Status: "
+ replica.replicaStatus()));
           }

           return response;

       } catch (ResourceNotFoundException e) {
           LOGGER.severe("Table not found: " + tableName + " - " +
e.getMessage());
           throw e;
       } catch (DynamoDbException e) {
           LOGGER.severe("Failed to describe table: " + tableName + " - " +
e.getMessage());
           throw e;
       }
```

```
    }
```

Ejemplo completo de operaciones de tablas globales mediante AWS SDK for Java 2.x.

```
   public static void exampleUsage(final Region sourceRegion, final Region
replicaRegion) {

       String tableName = "Music";
       String indexName = "SongTitleIndex";
       Long readCapacity = 15L;

       // Create DynamoDB client for the source region
       try (DynamoDbClient dynamoDbClient =
           DynamoDbClient.builder().region(sourceRegion).build()) {

           try {
               // Step 1: Create the initial table with GSI and streams
               LOGGER.info("Step 1: Creating table in source region: " +
sourceRegion.id());
               createTableWithGSI(dynamoDbClient, tableName, indexName);

               // Step 2: Wait for table to become active
               LOGGER.info("Step 2: Waiting for table to become active");
               waitForTableActive(dynamoDbClient, tableName);

               // Step 3: Add replica in destination region
               LOGGER.info("Step 3: Adding replica in region: " +
replicaRegion.id());
               addReplica(dynamoDbClient, tableName, replicaRegion, indexName,
readCapacity);

               // Step 4: Wait a moment for replica creation to start
               Thread.sleep(5000);

               // Step 5: Describe table to view replica information
               LOGGER.info("Step 5: Describing table to view replicas");
               describeTable(dynamoDbClient, tableName);

               // Step 6: Add a test item to verify replication
               LOGGER.info("Step 6: Adding test item to verify replication");
               putTestItem(dynamoDbClient, tableName, "TestArtist", "TestSong");
```

```
                LOGGER.info("Global table setup completed successfully!");
                LOGGER.info("You can verify replication by checking the item in
region: " + replicaRegion.id());

                // Step 7: Remove replica and clean up table
                LOGGER.info("Step 7: Removing replica from region: " +
replicaRegion.id());
                removeReplica(dynamoDbClient, tableName, replicaRegion);
                DeleteTableResponse deleteTableResponse =
dynamoDbClient.deleteTable(
                    DeleteTableRequest.builder().tableName(tableName).build());
                LOGGER.info("MREC global table demonstration completed
successfully!");

            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                throw new RuntimeException("Thread was interrupted", e);
            } catch (DynamoDbException e) {
                LOGGER.severe("DynamoDB operation failed: " + e.getMessage());
                throw e;
            }
        }
    }
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for Java 2.x.
  - CreateTable
  - DescribeTable
  - PutItem
  - UpdateTable

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Eliminación de datos de DynamoDB mediante instrucciones DELETE de PartiQL con un AWS SDK

En el siguiente ejemplo de código se muestra cómo eliminar datos mediante instrucciones DELETE de PartiQL.

JavaScript

SDK para JavaScript (v3)

Elimine elementos de una tabla de DynamoDB mediante instrucciones DELETE de PartiQL con AWS SDK para JavaScript.

```
/**
 * This example demonstrates how to delete items from a DynamoDB table using
 PartiQL.
 * It shows different ways to delete documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Delete a single item by its partition key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByPartitionKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
    Parameters: [partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item:", err);
    throw err;
  }
};

/**
 * Delete an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${sortKeyName} = ?`,
    Parameters: [partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted successfully");
```

```
    return data;
  } catch (err) {
    console.error("Error deleting item:", err);
    throw err;
  }
};

/**
 * Delete an item with a condition to ensure the delete only happens if a
 condition is met.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param conditionAttribute - The attribute to check in the condition
 * @param conditionValue - The value to compare against in the condition
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  conditionAttribute: string,
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${conditionAttribute} = ?`,
    Parameters: [partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted with condition successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item with condition:", err);
    throw err;
  }
};
```

```
/**
 * Batch delete multiple items using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param keys - Array of objects containing key information
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchDeleteItems = async (
  tableName: string,
  keys: Array<{
    partitionKeyName: string;
    partitionKeyValue: string | number;
    sortKeyName?: string;
    sortKeyValue?: string | number;
  }>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each delete
  const statements = keys.map((key) => {
    if (key.sortKeyName && key.sortKeyValue !== undefined) {
      return {
        Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ?
 AND ${key.sortKeyName} = ?`,
        Parameters: [key.partitionKeyValue, key.sortKeyValue],
      };
    } else {
      return {
        Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ?
`,
        Parameters: [key.partitionKeyValue],
      };
    }
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch deleted successfully");
    return data;
```

```
  } catch (err) {
    console.error("Error batch deleting items:", err);
    throw err;
  }
};

/**
 * Delete multiple items that match a filter condition.
 * Note: This performs a scan operation which can be expensive on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemsByFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items deleted by filter successfully");
    return data;
  } catch (err) {
    console.error("Error deleting items by filter:", err);
    throw err;
  }
};

/**
 * Example usage showing how to delete items with different index types
 */
export const deleteExamples = async () => {
  // Delete an item by partition key (simple primary key)
  await deleteItemByPartitionKey("UsersTable", "userId", "user123");
```

```
  // Delete an item by composite key (partition key + sort key)
  await deleteItemByCompositeKey(
    "OrdersTable",
    "orderId",
    "order456",
    "productId",
    "prod789"
  );

  // Delete with a condition
  await deleteItemWithCondition(
    "UsersTable",
    "userId",
    "user789",
    "userStatus",
    "inactive"
  );

  // Batch delete multiple items
  await batchDeleteItems("UsersTable", [
    { partitionKeyName: "userId", partitionKeyValue: "user234" },
    { partitionKeyName: "userId", partitionKeyValue: "user345" },
  ]);

  // Batch delete items with composite keys
  await batchDeleteItems("OrdersTable", [
    {
      partitionKeyName: "orderId",
      partitionKeyValue: "order567",
      sortKeyName: "productId",
      sortKeyValue: "prod123",
    },
    {
      partitionKeyName: "orderId",
      partitionKeyValue: "order678",
      sortKeyName: "productId",
      sortKeyValue: "prod456",
    },
  ]);

  // Delete items by filter (use with caution)
  await deleteItemsByFilter("UsersTable", "userStatus", "deleted");
};
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para JavaScript.
  - BatchExecuteStatement
  - ExecuteStatement

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Detección de EPI en imágenes con Amazon Rekognition mediante un AWS SDK

El siguiente ejemplo de código muestra cómo crear una aplicación que utiliza Amazon Rekognition para detectar equipos de protección individual (EPI) en imágenes.

Java

SDK para Java 2.x

Muestra cómo crear una función de AWS Lambda que detecte imágenes con equipos de protección individual.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- DynamoDB
- Amazon Rekognition
- Amazon S3
- Amazon SES

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Inserción de datos de DynamoDB mediante instrucciones INSERT de PartiQL con un AWS SDK

En el siguiente ejemplo de código se muestra cómo insertar datos mediante instrucciones INSERT de PartiQL.

JavaScript

SDK para JavaScript (v3)

Inserte elementos en una tabla de DynamoDB mediante instrucciones INSERT de PartiQL con AWS SDK para JavaScript.

```
/**
 * This example demonstrates how to insert items into a DynamoDB table using
 PartiQL.
 * It shows different ways to insert documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Insert a single item into a DynamoDB table using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @returns The response from the ExecuteStatementCommand
 */
export const insertItem = async (tableName: string, item: Record<string, any>) =>
 {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Convert the item to a string representation for PartiQL
  const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');

  const params = {
```

```
    Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item inserted successfully");
    return data;
  } catch (err) {
    console.error("Error inserting item:", err);
    throw err;
  }
};

/**
 * Insert multiple items into a DynamoDB table using PartiQL batch operation.
 * This is more efficient than inserting items one by one.
 *
 * @param tableName - The name of the DynamoDB table
 * @param items - Array of items to insert
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchInsertItems = async (tableName: string, items: Record<string,
 any>[]) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each item
  const statements = items.map((item) => {
    const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');
    return {
      Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
    };
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items inserted successfully");
    return data;
  } catch (err) {
    console.error("Error batch inserting items:", err);
```

```
    throw err;
  }
};

/**
 * Insert an item with a condition to prevent overwriting existing items.
 * This is useful for ensuring you don't accidentally overwrite data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @param partitionKeyName - The name of the partition key attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const insertItemWithCondition = async (
  tableName: string,
  item: Record<string, any>,
  partitionKeyName: string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const itemString = JSON.stringify(item).replace(/"([^"]+)":/g, '$1:');
  const partitionKeyValue = JSON.stringify(item[partitionKeyName]);

  const params = {
    Statement: `INSERT INTO "${tableName}" VALUE ${itemString} WHERE
 attribute_not_exists(${partitionKeyName})`,
    Parameters: [{ S: partitionKeyValue }],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item inserted with condition successfully");
    return data;
  } catch (err) {
    console.error("Error inserting item with condition:", err);
    throw err;
  }
};

/**
 * Example usage showing how to insert items with different index types
 */
export const insertExamples = async () => {
```

```
// Example table with a simple primary key (just partition key)
const simpleKeyItem = {
  userId: "user123",
  name: "John Doe",
  email: "john@example.com",
};
await insertItem("UsersTable", simpleKeyItem);

// Example table with composite key (partition key + sort key)
const compositeKeyItem = {
  orderId: "order456",
  productId: "prod789",
  quantity: 2,
  price: 29.99,
};
await insertItem("OrdersTable", compositeKeyItem);

// Example with Global Secondary Index (GSI)
// The GSI might be on the email attribute
const gsiItem = {
  userId: "user789",
  email: "jane@example.com",
  name: "Jane Smith",
  userType: "premium",  // This could be part of a GSI
};
await insertItem("UsersTable", gsiItem);

// Example with Local Secondary Index (LSI)
// LSI uses the same partition key but different sort key
const lsiItem = {
  orderId: "order567",  // Partition key
  productId: "prod123", // Sort key for the table
  orderDate: "2023-11-15", // Potential sort key for an LSI
  quantity: 1,
  price: 19.99,
};
await insertItem("OrdersTable", lsiItem);

// Batch insert example with multiple items
const batchItems = [
  {
    userId: "user234",
    name: "Alice Johnson",
    email: "alice@example.com",
```

```
    },
    {
      userId: "user345",
      name: "Bob Williams",
      email: "bob@example.com",
    },
  ];
  await batchInsertItems("UsersTable", batchItems);
};
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para JavaScript.
  - BatchExecuteStatement
  - ExecuteStatement

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Invocación de una función Lambda desde un navegador

En el siguiente ejemplo se muestra cómo invocar una función AWS Lambda desde un navegador.

JavaScript

SDK para JavaScript (v2)

Puede crear una aplicación basada en el navegador que utilice una función AWS Lambda para actualizar una tabla de Amazon DynamoDB con las selecciones del usuario.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- DynamoDB
- Lambda

SDK para JavaScript (v3)

Puede crear una aplicación basada en el navegador que utilice una función AWS Lambda para actualizar una tabla de Amazon DynamoDB con las selecciones del usuario. Esta aplicación utiliza AWS SDK para JavaScript v3.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- DynamoDB
- Lambda

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Administración de índices secundarios globales de DynamoDB mediante AWS Command Line Interface v2

En el siguiente ejemplo de código se muestra cómo administrar el ciclo de vida completo de los índices secundarios globales.

- Cree una tabla con un índice secundario global.
- Agregue un nuevo GSI a una tabla existente.
- Actualice (aumente) el rendimiento en caliente del GSI.
- Consulte datos mediante GSI.
- Elimine un GSI.

Bash

AWS CLI con Bash script

Cree una tabla con un índice secundario global.

```
# Create a table with a GSI
aws dynamodb create-table \
```

```
    --table-name MusicCollection \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
        AttributeName=AlbumTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --global-secondary-indexes \
        "IndexName=AlbumIndex,\
        KeySchema=[{AttributeName=AlbumTitle,KeyType=HASH}],\
        Projection={ProjectionType=ALL}"
```

Agregue un nuevo GSI (bajo demanda) a una tabla existente.

```
# Add a new GSI to an existing table
aws dynamodb update-table \
    --table-name MusicCollection \
    --attribute-definitions \
        AttributeName=Genre,AttributeType=S \
    --global-secondary-index-updates \
        "[{\"Create\":{\"IndexName\":\"GenreIndex\",\
        \"KeySchema\":[{\"AttributeName\":\"Genre\",\"KeyType\":\"HASH\"}],\
        \"Projection\":{\"ProjectionType\":\"ALL\"}}}]"
```

Actualice (aumente) el rendimiento en caliente del GSI.

```
# Increase the warm throughput of a GSI (default values are 12k reads, 4k writes)
aws dynamodb update-table \
    --table-name MusicCollection \
    --global-secondary-index-updates \
        "[{\"Update\":{\"IndexName\":\"AlbumIndex\",\
        \"WarmThroughput\":{\"ReadUnitsPerSecond\":15000,\"WriteUnitsPerSecond
\":6000}}}]"
```

Consulte datos mediante GSI.

```
# Query the AlbumIndex GSI
```

```
aws dynamodb query \
    --table-name MusicCollection \
    --index-name AlbumIndex \
    --key-condition-expression "AlbumTitle = :album" \
    --expression-attribute-values '{":album":{"S":"Let It Be"}}'

# Query the GenreIndex GSI
aws dynamodb query \
    --table-name MusicCollection \
    --index-name GenreIndex \
    --key-condition-expression "Genre = :genre" \
    --expression-attribute-values '{":genre":{"S":"Jazz"}}'
```

Elimine un GSI.

```
# Delete a GSI from a table
aws dynamodb update-table \
    --table-name MusicCollection \
    --global-secondary-index-updates \
        "[{\"Delete\":{\"IndexName\":\"GenreIndex\"}}]"
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de comandos de AWS CLI.
  - CreateTable
  - DeleteTable
  - Query
  - UpdateTable

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Administración de las políticas basadas en recursos de DynamoDB mediante AWS Command Line Interface v2

En el siguiente ejemplo de código se muestra cómo administrar el ciclo de vida completo de las políticas basadas en recursos para las tablas de DynamoDB.

- Cree una tabla con una política de recursos.
- Obtenga una política de recursos.
- Actualice una política de recursos.
- Elimine una política de recursos.

Bash

AWS CLI con Bash script

Cree una tabla con una política de recursos.

```
# Step 1: Create a DynamoDB table
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST

# Step 2: Create a resource-based policy document
cat > policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/DynamoDBReadOnly"
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
    }
  ]
}
```

```
EOF

# Step 3: Attach the resource-based policy to the table
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
 \
    --policy file://policy.json
```

Obtenga una política de recursos.

```
# Get the resource-based policy attached to a table
aws dynamodb get-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

Actualice una política de recursos.

```
# Step 1: Create an updated policy document
cat > updated-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::123456789012:role/DynamoDBReadOnly",
          "arn:aws:iam::123456789012:role/DynamoDBAnalytics"
        ]
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:Scan"
      ],
      "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
    }
  ]
}
EOF
```

```
# Step 2: Update the resource-based policy on the table
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
 \
    --policy file://updated-policy.json
```

Elimine una política de recursos.

```
# Delete the resource-based policy from a table
aws dynamodb delete-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de comandos de AWS CLI.
  - CreateTable
  - DeleteResourcePolicy
  - GetResourcePolicy
  - PutResourcePolicy

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Monitoreo del rendimiento de Amazon DynamoDB mediante un SDK de AWS

En el siguiente ejemplo se muestra cómo configurar el uso de DynamoDB en una aplicación para supervisar el rendimiento.

Java

SDK para Java 2.x

En este ejemplo se muestra cómo configurar una aplicación Java para supervisar el rendimiento de DynamoDB. La aplicación envía los datos de las métricas a CloudWatch, donde usted puede supervisar el rendimiento.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- CloudWatch
- DynamoDB

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Realización de operaciones de consulta avanzadas de DynamoDB mediante un SDK de AWS

En los siguientes ejemplos de código se muestra cómo realizar operaciones de consulta avanzadas en DynamoDB.

- Consulte tablas mediante diversas técnicas de filtrado y condición.
- Implemente la paginación para conjuntos de resultados grandes.
- Utilice índices secundarios globales para patrones de acceso alternativos.
- Aplique controles de coherencia en función de los requisitos de la aplicación.

Java

SDK para Java 2.x

Consulte con lecturas altamente coherentes mediante AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;
```

```
   public QueryResponse queryWithConsistentReads(
       final String tableName,
       final String partitionKeyName,
       final String partitionKeyValue,
       final boolean useConsistentRead) {

       CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);

       // Create expression attribute names for the column names
       final Map<String, String> expressionAttributeNames = new HashMap<>();
       expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);

       // Create expression attribute values for the column values
       final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
       expressionAttributeValues.put(
           EXPRESSION_ATTRIBUTE_VALUE_PK,
           AttributeValue.builder().s(partitionKeyValue).build());

       // Create the query request
       final QueryRequest queryRequest = QueryRequest.builder()
           .tableName(tableName)
           .keyConditionExpression(KEY_CONDITION_EXPRESSION)
           .expressionAttributeNames(expressionAttributeNames)
           .expressionAttributeValues(expressionAttributeValues)
           .consistentRead(useConsistentRead)
           .build();

       try {
           final QueryResponse response = dynamoDbClient.query(queryRequest);
           LOGGER.log(Level.INFO, "Query successful. Found {0} items",
response.count());
           return response;
       } catch (ResourceNotFoundException e) {
           LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
           throw e;
       } catch (DynamoDbException e) {
           LOGGER.log(Level.SEVERE, "Error querying with consistent reads", e);
           throw e;
       }
   }
```

Consulte mediante un índice secundario global con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryTable(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
```

```
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on base table successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw new DynamoDbQueryException("Table not found: " + tableName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying base table: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on base
table", e);
        }
    }

    /**
     * Queries a DynamoDB Global Secondary Index (GSI) by partition key.
     *
     * @param tableName         The name of the DynamoDB table
     * @param indexName         The name of the GSI
     * @param partitionKeyName  The name of the GSI partition key attribute
     * @param partitionKeyValue The value of the GSI partition key to query
     * @return The query response from DynamoDB
     * @throws ResourceNotFoundException if the table or index doesn't exist
     * @throws DynamoDbException if the query fails
     */
    public QueryResponse queryGlobalSecondaryIndex(
        final String tableName, final String indexName, final String
partitionKeyName, final String partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Index name", indexName);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_IK,
partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_IK,
```

```
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .indexName(indexName)
            .keyConditionExpression(GSI_KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on GSI successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(
                "Error: The Amazon DynamoDB table \"%s\" or index \"%s\" can't be
found.\n", tableName, indexName);
            throw new DynamoDbQueryException("Table or index not found: " +
tableName + "/" + indexName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying GSI: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on GSI",
e);
        }
    }
```

Consulte con paginación mediante AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
```

```
import java.util.Map;

    public List<Map<String, AttributeValue>> queryWithPagination(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue, final int pageSize) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validatePositiveInteger("Page size", pageSize);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(pageSize);

        // List to store all items from all pages
        final List<Map<String, AttributeValue>> allItems = new ArrayList<>();

        // Map to store the last evaluated key for pagination
        Map<String, AttributeValue> lastEvaluatedKey = null;
        int pageNumber = 1;

        try {
            do {
                // If we have a last evaluated key, use it for the next page
                if (lastEvaluatedKey != null) {
                    queryRequestBuilder.exclusiveStartKey(lastEvaluatedKey);
                }
```

```
                // Execute the query
                final QueryResponse response =
dynamoDbClient.query(queryRequestBuilder.build());

                // Process the current page of results
                final List<Map<String, AttributeValue>> pageItems =
response.items();
                allItems.addAll(pageItems);

                // Get the last evaluated key for the next page
                lastEvaluatedKey = response.lastEvaluatedKey();
                if (lastEvaluatedKey != null && lastEvaluatedKey.isEmpty()) {
                    lastEvaluatedKey = null;
                }

                System.out.println("Page " + pageNumber + ": Retrieved " +
pageItems.size() + " items (Running total: "
                    + allItems.size() + ")");

                pageNumber++;

            } while (lastEvaluatedKey != null);

            System.out.println("Query with pagination complete. Retrieved a total
of " + allItems.size()
                + " items across " + (pageNumber - 1) + " pages");

            return allItems;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with pagination: " +
e.getMessage());
            throw e;
        }
    }
```

Consulte con filtros complejos mediante AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithComplexFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String statusAttrName,
        final String activeStatus,
        final String pendingStatus,
        final String priceAttrName,
        final double minPrice,
        final double maxPrice,
        final String categoryAttrName) {

        // Validate parameters
        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Status attribute name",
 statusAttrName);
        CodeSampleUtils.validateStringParameter("Active status", activeStatus);
        CodeSampleUtils.validateStringParameter("Pending status", pendingStatus);
        CodeSampleUtils.validateStringParameter("Price attribute name",
 priceAttrName);
        CodeSampleUtils.validateStringParameter("Category attribute name",
 categoryAttrName);
        CodeSampleUtils.validateNumericRange("Minimum price", minPrice, 0.0,
 Double.MAX_VALUE);
        CodeSampleUtils.validateNumericRange("Maximum price", maxPrice, minPrice,
 Double.MAX_VALUE);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pk", partitionKeyName);
```

```
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_STATUS,
statusAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PRICE,
priceAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_CATEGORY,
categoryAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            ":pkValue", AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_ACTIVE,
            AttributeValue.builder().s(activeStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PENDING,
            AttributeValue.builder().s(pendingStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MIN_PRICE,
            AttributeValue.builder().n(String.valueOf(minPrice)).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MAX_PRICE,
            AttributeValue.builder().n(String.valueOf(maxPrice)).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        return dynamoDbClient.query(queryRequest);
    }
```

Consulte con una expresión de filtro construida dinámicamente mediante AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
```

```
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public static QueryResponse queryWithDynamicFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final Map<String, Object> filterCriteria,
        final Region region,
        final DynamoDbClient dynamoDbClient) {

        validateParameters(tableName, partitionKeyName, partitionKeyValue,
 filterCriteria);

        DynamoDbClient ddbClient = dynamoDbClient;
        boolean shouldClose = false;

        try {
            if (ddbClient == null) {
                ddbClient = createClient(region);
                shouldClose = true;
            }

            final QueryWithDynamicFilter queryHelper = new
 QueryWithDynamicFilter(ddbClient);
            return queryHelper.queryWithDynamicFilter(tableName,
 partitionKeyName, partitionKeyValue, filterCriteria);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Failed to execute dynamic filter query: " +
 e.getMessage());
            throw e;
        } catch (Exception e) {
            System.err.println("Unexpected error during query: " +
 e.getMessage());
            throw e;
```

```
        } finally {
            if (shouldClose && ddbClient != null) {
                ddbClient.close();
            }
        }
    }

    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue>
<filterAttrName> <filterAttrValue> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    filterAttrName - The name of the attribute to filter on.
                    filterAttrValue - The value to filter by.
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 5) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String partitionKeyName = args[1];
        final String partitionKeyValue = args[2];
        final String filterAttrName = args[3];
        final String filterAttrValue = args[4];
        final Region region = args.length > 5 ? Region.of(args[5]) :
Region.US_EAST_1;

        System.out.println("Querying items with dynamic filter: " +
filterAttrName + " = " + filterAttrValue);

        try {
            // Using the builder pattern to create and execute the query
            final QueryResponse response = new DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
```

```
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriterion(filterAttrName, filterAttrValue)
                .withRegion(region)
                .execute();

            // Process the results
            System.out.println("Found " + response.count() + " items:");
            response.items().forEach(item -> System.out.println(item));

            // Demonstrate multiple filter criteria
            System.out.println("\nNow querying with multiple filter criteria:");

            Map<String, Object> multipleFilters = new HashMap<>();
            multipleFilters.put(filterAttrName, filterAttrValue);
            multipleFilters.put("status", "active");

            final QueryResponse multiFilterResponse = new
DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriteria(multipleFilters)
                .withRegion(region)
                .execute();

            System.out.println("Found " + multiFilterResponse.count() + " items
with multiple filters:");
            multiFilterResponse.items().forEach(item ->
System.out.println(item));

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

Consulte con una expresión de filtro y limite mediante AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithFilterAndLimit(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String filterAttrName,
        final String filterAttrValue,
        final int limit) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Filter attribute name",
 filterAttrName);
        CodeSampleUtils.validateStringParameter("Filter attribute value",
 filterAttrValue);
        CodeSampleUtils.validatePositiveInteger("Limit", limit);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_FILTER,
 filterAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
```

```
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_FILTER,
            AttributeValue.builder().s(filterAttrValue).build());

        // Create the filter expression
        final String filterExpression = "#filterAttr = :filterValue";

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(filterExpression)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(limit)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query with filter and limit successful. Found
{0} items", response.count());
            LOGGER.log(
                Level.INFO, "ScannedCount: {0} (total items evaluated before
filtering)", response.scannedCount());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with filter and limit: {0}",
e.getMessage());
            throw e;
        }
    }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte con lecturas altamente coherentes mediante AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
      },
      ConsistentRead: useConsistentRead
    };

    // Execute the query
    const command = new QueryCommand(input);
```

```
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
  }
}
```

Consulte mediante un índice secundario global con AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
  config,
  tableName,
  userId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
    const input = {
      TableName: tableName,
      KeyConditionExpression: "user_id = :userId",
      ExpressionAttributeValues: {
        ":userId": { S: userId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying table: ${error}`);
```

```
    throw error;
  }
}

/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
  config,
  tableName,
  indexName,
  gameId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the GSI
    const input = {
      TableName: tableName,
      IndexName: indexName,
      KeyConditionExpression: "game_id = :gameId",
      ExpressionAttributeValues: {
        ":gameId": { S: gameId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying GSI: ${error}`);
    throw error;
  }
}
```

Consulte con paginación mediante AWS SDK para JavaScript.

```
/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  pageSize = 25
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize variables for pagination
    let lastEvaluatedKey = undefined;
    const allItems = [];
    let pageCount = 0;

    // Loop until all pages are retrieved
    do {
      // Construct the query input
      const input = {
```

```
        TableName: tableName,
        KeyConditionExpression: "#pk = :pkValue",
        Limit: pageSize,
        ExpressionAttributeNames: {
          "#pk": partitionKeyName
        },
        ExpressionAttributeValues: {
          ":pkValue": { S: partitionKeyValue }
        }
      };

      // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous
 query
      if (lastEvaluatedKey) {
        input.ExclusiveStartKey = lastEvaluatedKey;
      }

      // Execute the query
      const command = new QueryCommand(input);
      const response = await client.send(command);

      // Process the current page of results
      pageCount++;
      console.log(`Processing page ${pageCount} with ${response.Items.length}
 items`);

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems.push(...response.Items);
      }

      // Get the LastEvaluatedKey for the next page
      lastEvaluatedKey = response.LastEvaluatedKey;

    } while (lastEvaluatedKey); // Continue until there are no more pages

    console.log(`Query complete. Retrieved ${allItems.length} items in
 ${pageCount} pages.`);
    return allItems;
  } catch (error) {
    console.error(`Error querying with pagination: ${error}`);
    throw error;
  }
}
```

```
/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to
 retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */

module.exports = { queryWithPagination };
```

Consulte con filtros complejos mediante AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with a complex filter expression
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number|string} minViews - Minimum number of views for filtering
 * @param {number|string} minReplies - Minimum number of replies for filtering
 * @param {string} requiredTag - Tag that must be present in the item's tags set
 * @returns {Promise<Object>} - The query response
 */
```

```
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
 contains(tags, :tag)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with complex filter: ${error}`);
    throw error;
  }
}
```

Consulte con una expresión de filtro construida dinámicamente mediante AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

async function queryWithDynamicFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
      ":skValue": { S: sortKeyValue }
    };
    const expressionAttributeNames = {
      "#pk": partitionKeyName,
      "#sk": sortKeyName
    };

    // Add status filter if provided
    if (filterParams.status) {
      filterExpressions.push("status = :status");
      expressionAttributeValues[":status"] = { S: filterParams.status };
    }

    // Add minimum views filter if provided
    if (filterParams.minViews !== undefined) {
      filterExpressions.push("views >= :minViews");
      expressionAttributeValues[":minViews"] = { N:
 filterParams.minViews.toString() };
    }

    // Add author filter if provided
    if (filterParams.author) {
      filterExpressions.push("author = :author");
      expressionAttributeValues[":author"] = { S: filterParams.author };
```

```
    }

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
    };

    // Add filter expression if any filters were provided
    if (filterExpressions.length > 0) {
      input.FilterExpression = filterExpressions.join(" AND ");
    }

    // Add expression attribute names and values
    input.ExpressionAttributeNames = expressionAttributeNames;
    input.ExpressionAttributeValues = expressionAttributeValues;

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with dynamic filter: ${error}`);
    throw error;
  }
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte con lecturas altamente coherentes mediante AWS SDK for Python (Boto3).

```
import time

import boto3
from boto3.dynamodb.conditions import Key


def query_with_consistent_read(
```

```
    table_name,
    partition_key_name,
    partition_key_value,
    sort_key_name=None,
    sort_key_value=None,
    consistent_read=True,
):
    """
    Query a DynamoDB table with the option for strongly consistent reads.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str, optional): The name of the sort key attribute.
        sort_key_value (str, optional): The value of the sort key to query.
        consistent_read (bool, optional): Whether to use strongly consistent
 reads. Defaults to True.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    if sort_key_name and sort_key_value:
        key_condition = key_condition & Key(sort_key_name).eq(sort_key_value)

    # Perform the query with the consistent read option
    response = table.query(KeyConditionExpression=key_condition,
 ConsistentRead=consistent_read)

    return response
```

Consulte mediante un índice secundario global con AWS SDK for Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Key


def query_table(table_name, partition_key_name, partition_key_value):
    """
    Query a DynamoDB table using its primary key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query on the table's primary key
    response =
 table.query(KeyConditionExpression=Key(partition_key_name).eq(partition_key_value))

    return response


def query_gsi(table_name, index_name, partition_key_name, partition_key_value):
    """
    Query a Global Secondary Index (GSI) on a DynamoDB table.

    Args:
        table_name (str): The name of the DynamoDB table.
        index_name (str): The name of the Global Secondary Index.
        partition_key_name (str): The name of the GSI's partition key attribute.
        partition_key_value (str): The value of the GSI's partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Perform the query on the GSI
    response = table.query(
        IndexName=index_name,
 KeyConditionExpression=Key(partition_key_name).eq(partition_key_value)
    )

    return response
```

Consulte con paginación mediante AWS SDK for Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Key


def query_with_pagination(
    table_name, partition_key_name, partition_key_value, page_size=25,
 max_pages=None
):
    """
    Query a DynamoDB table with pagination to handle large result sets.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
 Defaults to 25.
        max_pages (int, optional): The maximum number of pages to retrieve. If
 None, retrieves all pages.

    Returns:
        list: All items retrieved from the query across all pages.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
```

```
    page_count = 0
    all_items = []

    # Paginate through the results
    while True:
        # Check if we've reached the maximum number of pages
        if max_pages is not None and page_count >= max_pages:
            break

        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
 Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
 query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Process the current page of results
        items = response.get("Items", [])
        all_items.extend(items)

        # Update pagination tracking
        page_count += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if not last_evaluated_key:
            break

    return all_items


def query_with_pagination_generator(
    table_name, partition_key_name, partition_key_value, page_size=25
):
```

```
    """
    Query a DynamoDB table with pagination using a generator to handle large
result sets.
    This approach is memory-efficient as it yields one page at a time.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
Defaults to 25.

    Yields:
        tuple: A tuple containing (items, page_number, last_page) where:
            - items is a list of items for the current page
            - page_number is the current page number (starting from 1)
            - last_page is a boolean indicating if this is the last page
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
    page_number = 0

    # Paginate through the results
    while True:
        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Get the current page of results
```

```
        items = response.get("Items", [])
        page_number += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # Determine if this is the last page
        is_last_page = last_evaluated_key is None

        # Yield the current page of results
        yield (items, page_number, is_last_page)

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if is_last_page:
            break
```

Consulte con filtros complejos mediante AWS SDK for Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_complex_filter(
    table_name,
    partition_key_name,
    partition_key_value,
    min_rating=None,
    status_list=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        min_rating (float, optional): Minimum rating value for filtering.
        status_list (list, optional): List of status values to include.
        max_price (float, optional): Maximum price value for filtering.
```

```
    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize the filter expression and expression attribute values
    filter_expression = None
    expression_attribute_values = {}

    # Build the filter expression based on provided parameters
    if min_rating is not None:
        filter_expression = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if status_list and len(status_list) > 0:
        status_condition = None
        for i, status in enumerate(status_list):
            status_value_name = f":status{i}"
            expression_attribute_values[status_value_name] = status

            if status_condition is None:
                status_condition = Attr("status").eq(status)
            else:
                status_condition = status_condition | Attr("status").eq(status)

        if filter_expression is None:
            filter_expression = status_condition
        else:
            filter_expression = filter_expression & status_condition

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if filter_expression is None:
            filter_expression = price_condition
        else:
            filter_expression = filter_expression & price_condition
```

```
    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
 expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response


def query_with_complex_filter_and_or(
    table_name,
    partition_key_name,
    partition_key_value,
    category=None,
    min_rating=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression using AND and OR
 operators.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        category (str, optional): Category value for filtering.
        min_rating (float, optional): Minimum rating value for filtering.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)
```

```
    # Build a complex filter expression with AND and OR operators
    filter_expression = None
    expression_attribute_values = {}

    # Build the category condition
    if category:
        filter_expression = Attr("category").eq(category)
        expression_attribute_values[":category"] = category

    # Build the rating and price condition (rating >= min_rating OR price <=
max_price)
    rating_price_condition = None

    if min_rating is not None:
        rating_price_condition = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if rating_price_condition is None:
            rating_price_condition = price_condition
        else:
            rating_price_condition = rating_price_condition | price_condition

    # Combine the conditions
    if rating_price_condition:
        if filter_expression is None:
            filter_expression = rating_price_condition
        else:
            filter_expression = filter_expression & rating_price_condition

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
expression_attribute_values

    # Execute the query
```

```
    response = table.query(**query_params)
    return response
```

Consulte con una expresión de filtro construida dinámicamente mediante AWS SDK for Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_dynamic_filter(
    table_name, partition_key_name, partition_key_value, filter_conditions=None
):
    """
    Query a DynamoDB table with a dynamically constructed filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_conditions (dict, optional): A dictionary of filter conditions
 where
            keys are attribute names and values are dictionaries with 'operator'
 and 'value'.
            Example: {'rating': {'operator': '>=', 'value': 4}, 'status':
 {'operator': '=', 'value': 'active'}}

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize variables for the filter expression and attribute values
    filter_expression = None
    expression_attribute_values = {":pk_val": partition_key_value}
```

```
    # Dynamically build the filter expression if filter conditions are provided
    if filter_conditions:
        for attr_name, condition in filter_conditions.items():
            operator = condition.get("operator")
            value = condition.get("value")
            attr_value_name = f":{attr_name}"
            expression_attribute_values[attr_value_name] = value

            # Create the appropriate filter expression based on the operator
            current_condition = None
            if operator == "=":
                current_condition = Attr(attr_name).eq(value)
            elif operator == "!=":
                current_condition = Attr(attr_name).ne(value)
            elif operator == ">":
                current_condition = Attr(attr_name).gt(value)
            elif operator == ">=":
                current_condition = Attr(attr_name).gte(value)
            elif operator == "<":
                current_condition = Attr(attr_name).lt(value)
            elif operator == "<=":
                current_condition = Attr(attr_name).lte(value)
            elif operator == "contains":
                current_condition = Attr(attr_name).contains(value)
            elif operator == "begins_with":
                current_condition = Attr(attr_name).begins_with(value)

            # Combine with existing filter expression using AND
            if current_condition:
                if filter_expression is None:
                    filter_expression = current_condition
                else:
                    filter_expression = filter_expression & current_condition

    # Perform the query with the dynamically built filter expression
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression

    response = table.query(**query_params)
    return response
```

Consulte con una expresión de filtro y limite mediante AWS SDK for Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_filter_and_limit(
    table_name,
    partition_key_name,
    partition_key_value,
    filter_attribute=None,
    filter_value=None,
    limit=10,
):
    """
    Query a DynamoDB table with a filter expression and limit the number of
 results.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_attribute (str, optional): The attribute name to filter on.
        filter_value (any, optional): The value to compare against in the filter.
        limit (int, optional): The maximum number of items to evaluate. Defaults
 to 10.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition, "Limit": limit}
```

```
    # Add the filter expression if filter attributes are provided
    if filter_attribute and filter_value is not None:
        query_params["FilterExpression"] =
Attr(filter_attribute).gt(filter_value)
        query_params["ExpressionAttributeValues"] = {":filter_value":
filter_value}

    # Execute the query
    response = table.query(**query_params)
    return response
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Realización de operaciones de lista en DynamoDB con un AWS SDK

En los siguientes ejemplos de código se muestra cómo realizar operaciones de lista en DynamoDB.

- Agregue elementos a un atributo de lista.
- Elimine elementos de un atributo de lista.
- Actualice elementos específicos de una lista por índice.
- Utilice las funciones de agregar a listas e indexar listas.

Java

SDK para Java 2.x

Demuestre las operaciones de lista con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
```

```
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    /**
     * Appends items to a list attribute.
     *
     * <p>This method demonstrates how to use the list_append function to add
     * items to the end of a list attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param itemsToAppend The items to append to the list
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse appendToList(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        List<AttributeValue> itemsToAppend) {

        // Create a list value from the items to append
        AttributeValue listValue =
 AttributeValue.builder().l(itemsToAppend).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attrName =
 list_append(if_not_exists(#attrName, :emptyList), :newItems)")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .expressionAttributeValues(Map.of(
                ":newItems",
                listValue,
```

```
                ":emptyList",
                AttributeValue.builder().l(new
ArrayList<AttributeValue>()).build()))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Prepends items to a list attribute.
     *
     * <p>This method demonstrates how to use the list_append function to add
     * items to the beginning of a list attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param itemsToPrepend The items to prepend to the list
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse prependToList(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        List<AttributeValue> itemsToPrepend) {

        // Create a list value from the items to prepend
        AttributeValue listValue =
AttributeValue.builder().l(itemsToPrepend).build();

        // Define the update parameters
        // Note: To prepend, we put the new items first in the list_append
function
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attrName = list_append(:newItems,
if_not_exists(#attrName, :emptyList))")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
```

```
            .expressionAttributeValues(Map.of(
                ":newItems",
                listValue,
                ":emptyList",
                AttributeValue.builder().l(new
ArrayList<AttributeValue>()).build()))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Updates a specific element in a list attribute.
     *
     * <p>This method demonstrates how to update a specific element in a list
     * by its index.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param index The index of the element to update
     * @param newValue The new value for the element
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateListElement(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        int index,
        AttributeValue newValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attrName[" + index + "] = :newValue")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .expressionAttributeValues(Map.of(":newValue", newValue))
            .returnValues("UPDATED_NEW")
```

```
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Removes a specific element from a list attribute.
     *
     * <p>This method demonstrates how to remove a specific element from a list
     * by its index.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param listAttributeName The name of the list attribute
     * @param index The index of the element to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse removeListElement(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String listAttributeName,
        int index) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("REMOVE #attrName[" + index + "]")
            .expressionAttributeNames(Map.of("#attrName", listAttributeName))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Gets the current value of a list attribute.
     *
     * <p>Helper method to retrieve the current value of a list attribute.
```

```
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param listAttributeName The name of the list attribute
     * @return The list attribute value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static List<AttributeValue> getListAttribute(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String listAttributeName) {

        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(listAttributeName)
            .build();

        try {
            // Perform the get operation
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the list attribute if it exists, otherwise null
            if (response.item() != null &&
response.item().containsKey(listAttributeName)) {
                return response.item().get(listAttributeName).l();
            }

            return null;
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get list attribute: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

Ejemplo de uso de operaciones de lista con AWS SDK for Java 2.x.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
```

```
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating list operations in DynamoDB");

        try {
            // Example 1: Append items to a list
            System.out.println("\nExample 1: Appending items to a list");
            List<AttributeValue> tagsToAppend = List.of(
                AttributeValue.builder().s("Electronics").build(),
                AttributeValue.builder().s("Gadget").build());

            UpdateItemResponse appendResponse = appendToList(dynamoDbClient,
tableName, key, "Tags", tagsToAppend);

            System.out.println("Updated list attribute: " +
appendResponse.attributes());

            // Example 2: Prepend items to a list
            System.out.println("\nExample 2: Prepending items to a list");
            List<AttributeValue> tagsToPrepend = List.of(
                AttributeValue.builder().s("Featured").build(),
                AttributeValue.builder().s("New").build());

            UpdateItemResponse prependResponse = prependToList(dynamoDbClient,
tableName, key, "Tags", tagsToPrepend);

            System.out.println("Updated list attribute: " +
prependResponse.attributes());

            // Example 3: Update a specific element in a list
            System.out.println("\nExample 3: Updating a specific element in a
list");
            UpdateItemResponse updateResponse = updateListElement(
                dynamoDbClient,
                tableName,
                key,
                "Tags",
                0,
                AttributeValue.builder().s("BestSeller").build());

            System.out.println("Updated list attribute: " +
updateResponse.attributes());
```

```
            // Example 4: Remove a specific element from a list
            System.out.println("\nExample 4: Removing a specific element from a
list");
            UpdateItemResponse removeResponse = removeListElement(dynamoDbClient,
tableName, key, "Tags", 1);

            System.out.println("Updated list attribute: " +
removeResponse.attributes());

            // Example 5: Get the current value of a list attribute
            System.out.println("\nExample 5: Getting the current value of a list
attribute");
            List<AttributeValue> currentList = getListAttribute(dynamoDbClient,
tableName, key, "Tags");

            if (currentList != null) {
                System.out.println("Current list attribute:");
                for (int i = 0; i < currentList.size(); i++) {
                    System.out.println("  [" + i + "]: " +
currentList.get(i).s());
                }
            } else {
                System.out.println("List attribute not found");
            }

            // Explain list operations
            System.out.println("\nKey points about DynamoDB list operations:");
            System.out.println("1. Lists are ordered collections of attributes");
            System.out.println("2. Use list_append to add items to a list");
            System.out.println("3. To append items, use list_append(existingList,
newItems)");
            System.out.println("4. To prepend items, use list_append(newItems,
existingList)");
            System.out.println("5. Use index notation (list[0]) to access or
update specific elements");
            System.out.println("6. Use REMOVE to delete elements from a list");
            System.out.println("7. List indices are zero-based");
            System.out.println("8. Use if_not_exists to handle the case where the
list doesn't exist yet");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
```

```
        }
    }
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Demuestre las operaciones de lista con AWS SDK para JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Append elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the end of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to append to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function appendToList(
  config,
  tableName,
  key,
  listName,
  values
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
```

```
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} =
 list_append(if_not_exists(${listName}, :empty_list), :values)`,
    ExpressionAttributeValues: {
      ":empty_list": [],
      ":values": values
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Prepend elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the beginning of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to prepend to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function prependToList(
  config,
  tableName,
  key,
  listName,
  values
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Define the update parameters using list_append
  // Note: To prepend, we put the new values first in the list_append function
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName} = list_append(:values,
 if_not_exists(${listName}, :empty_list))`,
    ExpressionAttributeValues: {
      ":empty_list": [],
      ":values": values
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update a specific element in a list by index.
 *
 * This function demonstrates how to update a specific element in a list
 * using the index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to update
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateListElement(
  config,
  tableName,
  key,
  listName,
  index,
  value
) {
  // Initialize the DynamoDB client
```

```
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using index notation
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${listName}[${index}] = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Remove an element from a list by index.
 *
 * This function demonstrates how to remove a specific element from a list
 * using the REMOVE action with index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to remove
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeListElement(
  config,
  tableName,
  key,
  listName,
  index
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Define the update parameters using REMOVE with index notation
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `REMOVE ${listName}[${index}]`,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Concatenate two lists.
 *
 * This function demonstrates how to concatenate two lists using the list_append
 function.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName1 - The name of the first list attribute
 * @param {string} listName2 - The name of the second list attribute
 * @param {string} resultListName - The name of the attribute to store the
 concatenated list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function concatenateLists(
  config,
  tableName,
  key,
  listName1,
  listName2,
  resultListName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  const params = {
    TableName: tableName,
```

```
    Key: key,
    UpdateExpression: `SET ${resultListName} =
 list_append(if_not_exists(${listName1}, :empty_list),
 if_not_exists(${listName2}, :empty_list))`,
    ExpressionAttributeValues: {
      ":empty_list": []
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Create a nested list structure.
 *
 * This function demonstrates how to create and work with nested lists.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} nestedLists - An array of arrays to create a nested list
 structure
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createNestedList(
  config,
  tableName,
  key,
  listName,
  nestedLists
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters to create a nested list
  const params = {
    TableName: tableName,
    Key: key,
```

```
    UpdateExpression: `SET ${listName} = :nested_lists`,
    ExpressionAttributeValues: {
      ":nested_lists": nestedLists
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update an element in a nested list.
 *
 * This function demonstrates how to update an element in a nested list
 * using multiple index notations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} outerIndex - The index in the outer list
 * @param {number} innerIndex - The index in the inner list
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateNestedListElement(
  config,
  tableName,
  key,
  listName,
  outerIndex,
  innerIndex,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using multiple index notations
  const params = {
    TableName: tableName,
```

```
    Key: key,
    UpdateExpression: `SET ${listName}[${outerIndex}][${innerIndex}] = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
```

```
}
```

Ejemplo de uso de operaciones de lista con AWS SDK para JavaScript.

```
/**
 * Example of how to work with lists in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "UserProfiles";
  const key = { UserId: "U12345" };

  console.log("Demonstrating list operations in DynamoDB");

  try {
    // Example 1: Append elements to a list
    console.log("\nExample 1: Appending elements to a list");
    const response1 = await appendToList(
      config,
      tableName,
      key,
      "RecentSearches",
      ["laptop", "headphones", "monitor"]
    );

    console.log("Appended to list:", response1.Attributes);

    // Example 2: Prepend elements to a list
    console.log("\nExample 2: Prepending elements to a list");
    const response2 = await prependToList(
      config,
      tableName,
      key,
      "RecentSearches",
      ["keyboard", "mouse"]
    );

    console.log("Prepended to list:", response2.Attributes);

    // Get the current state of the item
    let currentItem = await getItem(config, tableName, key);
```

```
    console.log("\nCurrent state of RecentSearches:",
currentItem?.RecentSearches);

    // Example 3: Update a specific element in a list
    console.log("\nExample 3: Updating a specific element in a list");
    const response3 = await updateListElement(
      config,
      tableName,
      key,
      "RecentSearches",
      0, // Update the first element
      "mechanical keyboard" // New value
    );

    console.log("Updated list element:", response3.Attributes);

    // Example 4: Remove an element from a list
    console.log("\nExample 4: Removing an element from a list");
    const response4 = await removeListElement(
      config,
      tableName,
      key,
      "RecentSearches",
      2 // Remove the third element
    );

    console.log("List after removing element:", response4.Attributes);

    // Example 5: Create and concatenate lists
    console.log("\nExample 5: Creating and concatenating lists");

    // First, create two separate lists
    await updateWithMultipleActions(
      config,
      tableName,
      key,
      "SET WishList = :wishlist, SavedItems = :saveditems",
      null,
      {
        ":wishlist": ["gaming laptop", "wireless earbuds"],
        ":saveditems": ["smartphone", "tablet"]
      }
    );
```

```
    // Then, concatenate them
    const response5 = await concatenateLists(
      config,
      tableName,
      key,
      "WishList",
      "SavedItems",
      "AllItems"
    );

    console.log("Concatenated lists:", response5.Attributes);

    // Example 6: Create a nested list structure
    console.log("\nExample 6: Creating a nested list structure");
    const response6 = await createNestedList(
      config,
      tableName,
      key,
      "Categories",
      [
        ["Electronics", "Computers", "Accessories"],
        ["Books", "Magazines", "E-books"],
        ["Clothing", "Shoes", "Watches"]
      ]
    );

    console.log("Created nested list:", response6.Attributes);

    // Example 7: Update an element in a nested list
    console.log("\nExample 7: Updating an element in a nested list");
    const response7 = await updateNestedListElement(
      config,
      tableName,
      key,
      "Categories",
      0, // First inner list
      1, // Second element in that list
      "Laptops" // New value
    );

    console.log("Updated nested list element:", response7.Attributes);

    // Get the final state of the item
    currentItem = await getItem(config, tableName, key);
```

```
    console.log("\nFinal state of the item:", JSON.stringify(currentItem, null,
 2));

    // Explain list operations
    console.log("\nKey points about list operations in DynamoDB:");
    console.log("1. Use list_append to add elements to a list");
    console.log("2. To append elements, use list_append(existingList,
 newElements)");
    console.log("3. To prepend elements, use list_append(newElements,
 existingList)");
    console.log("4. Use if_not_exists to handle cases where the list might not
 exist yet");
    console.log("5. Use index notation (list[0]) to access or update specific
 elements");
    console.log("6. Use REMOVE with index notation to remove elements from a
 list");
    console.log("7. Lists can contain elements of different types");
    console.log("8. Lists can be nested (lists of lists)");
    console.log("9. Use multiple index notations (list[0][1]) to access nested
 list elements");

  } catch (error) {
    console.error("Error:", error);
  }
}

/**
 * Helper function for the examples.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} updateExpression - The update expression
 * @param {Object} expressionAttributeNames - Expression attribute name
 placeholders
 * @param {Object} expressionAttributeValues - Expression attribute value
 placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
  config,
  tableName,
  key,
  updateExpression,
```

```
  expressionAttributeNames,
  expressionAttributeValues
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Prepare the update parameters
  const updateParams = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ReturnValues: "UPDATED_NEW"
  };

  // Add expression attribute names if provided
  if (expressionAttributeNames) {
    updateParams.ExpressionAttributeNames = expressionAttributeNames;
  }

  // Add expression attribute values if provided
  if (expressionAttributeValues) {
    updateParams.ExpressionAttributeValues = expressionAttributeValues;
  }

  // Execute the update
  const response = await docClient.send(new UpdateCommand(updateParams));

  return response;
}
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Demuestre las operaciones de lista con AWS SDK for Python (Boto3).

```
import boto3
import json
```

```
from typing import Any, Dict, List, Optional, Union


def create_list_attribute(
    table_name: str, key: Dict[str, Any], list_name: str, list_values: List[Any]
) -> Dict[str, Any]:
    """
    Create a new list attribute or replace an existing one.

    This function demonstrates how to create a new list attribute or replace
    an existing list with new values.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        list_values (List[Any]): The values to set in the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the SET operation to create or replace the list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} = :list_values",
        ExpressionAttributeValues={":list_values": list_values},
        ReturnValues="UPDATED_NEW",
    )

    return response


def append_to_list(
    table_name: str, key: Dict[str, Any], list_name: str, values_to_append:
 List[Any]
) -> Dict[str, Any]:
    """
    Append values to the end of a list attribute.
```

```
    This function demonstrates how to use the list_append function to add
 elements
    to the end of a list attribute.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        values_to_append (List[Any]): The values to append to the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use list_append to add values to the end of the list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} = list_append({list_name}, :values)",
        ExpressionAttributeValues={":values": values_to_append},
        ReturnValues="UPDATED_NEW",
    )

    return response


def prepend_to_list(
    table_name: str, key: Dict[str, Any], list_name: str, values_to_prepend:
 List[Any]
) -> Dict[str, Any]:
    """
    Prepend values to the beginning of a list attribute.

    This function demonstrates how to use the list_append function to add
 elements
    to the beginning of a list attribute.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
```

```
        values_to_prepend (List[Any]): The values to prepend to the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use list_append with reversed order to add values to the beginning of the
 list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} = list_append(:values, {list_name})",
        ExpressionAttributeValues={":values": values_to_prepend},
        ReturnValues="UPDATED_NEW",
    )

    return response


def update_list_element(
    table_name: str, key: Dict[str, Any], list_name: str, index: int, new_value:
 Any
) -> Dict[str, Any]:
    """
    Update a specific element in a list attribute.

    This function demonstrates how to update a specific element in a list
 attribute
    using the index notation.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        index (int): The zero-based index of the element to update.
        new_value (Any): The new value for the element.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
```

```
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the index notation to update a specific element
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name}[{index}] = :value",
        ExpressionAttributeValues={":value": new_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def remove_list_element(
    table_name: str, key: Dict[str, Any], list_name: str, index: int
) -> Dict[str, Any]:
    """
    Remove a specific element from a list attribute.

    This function demonstrates how to remove a specific element from a list
 attribute
    using the REMOVE action with index notation.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        index (int): The zero-based index of the element to remove.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the REMOVE action with index notation to remove a specific element
    response = table.update_item(
        Key=key, UpdateExpression=f"REMOVE {list_name}[{index}]",
 ReturnValues="UPDATED_NEW"
    )
```

```
    return response


def update_nested_list_element(
    table_name: str, key: Dict[str, Any], path: str, new_value: Any
) -> Dict[str, Any]:
    """
    Update an element in a nested list structure.

    This function demonstrates how to update an element in a nested list
 structure
    using expression attribute names for the path components.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        path (str): The path to the nested element (e.g., "parent[0].child[1]").
        new_value (Any): The new value for the element.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Define a type for path parts
    path_part = Dict[str, Union[str, int]]
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Parse the path to extract attribute names and indices
    path_parts: List[path_part] = []
    current_part = ""
    in_bracket = False

    for char in path:
        if char == "[":
            if current_part:
                path_parts.append({"type": "attribute", "value": current_part})
                current_part = ""
            in_bracket = True
        elif char == "]":
            if current_part:
                # Fix for mypy: Use a properly typed dictionary with Union type
```

```
                path_parts.append({"type": "index", "value": int(current_part)})
                current_part = ""
            in_bracket = False
        elif char == "." and not in_bracket:
            if current_part:
                path_parts.append({"type": "attribute", "value": current_part})
                current_part = ""
        else:
            current_part += char

    if current_part:
        path_parts.append({"type": "attribute", "value": current_part})

    # Build the update expression and attribute names
    update_expression = "SET "
    expression_attribute_names = {}

    # Build the path expression
    path_expression = ""
    for i, part in enumerate(path_parts):
        if part["type"] == "attribute":
            name_placeholder = f"#attr{i}"
            expression_attribute_names[name_placeholder] = part["value"]

            if path_expression:
                path_expression += "."
            path_expression += name_placeholder
        elif part["type"] == "index":
            path_expression += f"[{part['value']}]"

    # Complete the update expression
    update_expression += f"{path_expression} = :value"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues={":value": new_value},
        ReturnValues="UPDATED_NEW",
    )

    return response
```

```
def create_list_if_not_exists(
    table_name: str, key: Dict[str, Any], list_name: str, default_values:
 List[Any]
) -> Dict[str, Any]:
    """
    Create a list attribute if it doesn't exist.

    This function demonstrates how to use if_not_exists to create a list
 attribute
    with default values if it doesn't already exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        default_values (List[Any]): The default values for the list.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use if_not_exists to create the list if it doesn't exist
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} =
 if_not_exists({list_name}, :default)",
        ExpressionAttributeValues={":default": default_values},
        ReturnValues="UPDATED_NEW",
    )

    return response


def append_to_list_safely(
    table_name: str,
    key: Dict[str, Any],
    list_name: str,
    values_to_append: List[Any],
    default_values: Optional[List[Any]] = None,
```

```
) -> Dict[str, Any]:
    """
    Append values to a list, creating it if it doesn't exist.

    This function demonstrates how to safely append values to a list attribute,
    creating the list with default values if it doesn't exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        list_name (str): The name of the list attribute.
        values_to_append (List[Any]): The values to append to the list.
        default_values (Optional[List[Any]]): The default values if the list
doesn't exist.
            If not provided, values_to_append will be used as the default.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # If default_values is not provided, use values_to_append
    if default_values is None:
        default_values = values_to_append

    # Use if_not_exists with list_append to safely append to the list
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {list_name} =
list_append(if_not_exists({list_name}, :default), :values)",
        ExpressionAttributeValues={
            ":default": default_values if default_values else [],
            ":values": values_to_append,
        },
        ReturnValues="UPDATED_NEW",
    )

    return response
```

Ejemplo de uso de operaciones de lista con AWS SDK for Python (Boto3).

```
def example_usage():
    """Example of how to use list operations in DynamoDB."""
    # Example parameters
    table_name = "UserData"
    key = {"UserId": "user123"}

    print("Example 1: Creating a list attribute")
    try:
        response = create_list_attribute(
            table_name=table_name,
            key=key,
            list_name="Interests",
            list_values=["Reading", "Hiking", "Photography"],
        )
        print(
            f"List attribute created successfully:
 {json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error creating list attribute: {e}")

    print("\nExample 2: Appending values to a list")
    try:
        response = append_to_list(
            table_name=table_name,
            key=key,
            list_name="Interests",
            values_to_append=["Cooking", "Gardening"],
        )
        print(
            f"Values appended to list successfully:
 {json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error appending to list: {e}")

    print("\nExample 3: Prepending values to a list")
    try:
        response = prepend_to_list(
```

```
            table_name=table_name,
            key=key,
            list_name="Interests",
            values_to_prepend=["Travel", "Music"],
        )
        print(
            f"Values prepended to list successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error prepending to list: {e}")

    print("\nExample 4: Updating a specific list element")
    try:
        response = update_list_element(
            table_name=table_name,
            key=key,
            list_name="Interests",
            index=2,
            new_value="Mountain Hiking",
        )
        print(
            f"List element updated successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error updating list element: {e}")

    print("\nExample 5: Removing a list element")
    try:
        response = remove_list_element(
            table_name=table_name, key=key, list_name="Interests", index=0
        )
        print(
            f"List element removed successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error removing list element: {e}")

    print("\nExample 6: Working with nested lists")
    try:
        # First, create an item with a nested structure
        dynamodb = boto3.resource("dynamodb")
```

```
        table = dynamodb.Table(table_name)

        table.update_item(
            Key={"UserId": "user456"},
            UpdateExpression="SET #skills = :skills",
            ExpressionAttributeNames={"#skills": "Skills"},
            ExpressionAttributeValues={
                ":skills": [
                    {"Category": "Programming", "Languages": ["Python", "Java",
"JavaScript"]},
                    {"Category": "Database", "Systems": ["DynamoDB", "MongoDB",
"PostgreSQL"]},
                ]
            },
        )

        # Now update a nested element
        response = update_nested_list_element(
            table_name=table_name,
            key={"UserId": "user456"},
            path="Skills[0].Languages[1]",
            new_value="TypeScript",
        )
        print(
            f"Nested list element updated successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error working with nested lists: {e}")

    print("\nExample 7: Creating a list if it doesn't exist")
    try:
        response = create_list_if_not_exists(
            table_name=table_name,
            key={"UserId": "user789"},
            list_name="Preferences",
            default_values=["Default1", "Default2", "Default3"],
        )
        print(
            f"List created with default values:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error creating list with default values: {e}")
```

```
    print("\nExample 8: Safely appending to a list")
    try:
        response = append_to_list_safely(
            table_name=table_name,
            key={"UserId": "user789"},
            list_name="Notifications",
            values_to_append=["New message received"],
            default_values=[],
        )
        print(f"Safely appended to list: {json.dumps(response.get('Attributes',
{}), default=str)}")
    except Exception as e:
        print(f"Error safely appending to list: {e}")

    print("\nKey Points About Working with Lists in DynamoDB:")
    print("1. Lists are ordered collections of elements that can be of different
types")
    print("2. Use the SET operation with direct assignment to create or replace a
list")
    print("3. Use list_append() to add elements to a list without replacing the
entire list")
    print("4. To append to the end: list_append(list_name, :values)")
    print("5. To prepend to the beginning: list_append(:values, list_name)")
    print("6. Use index notation list_name[index] to access or update specific
elements")
    print("7. Use the REMOVE action with index notation to remove specific
elements")
    print("8. Lists can contain nested structures like maps and other lists")
    print("9. Use if_not_exists() to create a list with default values if it
doesn't exist")
    print("10. List indices are zero-based (the first element is at index 0)")
    print("11. Attempting to access an index beyond the list bounds will result
in an error")
```

- Para obtener información la API, consulte UpdateItem en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Realización de operaciones de asignación en DynamoDB con un AWS SDK

En los siguientes ejemplos de código se muestra cómo realizar operaciones de asignación en DynamoDB.

- Agregue y actualice atributos anidados en estructuras de asignación.
- Elimine campos específicos de asignaciones.
- Trabaje con atributos de asignación profundamente anidados.

Java

SDK para Java 2.x

Demuestre las operaciones de asignación con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

    /**
     * Updates a map attribute that may not exist.
     *
     * <p>This method demonstrates how to safely update a map attribute
     * by using if_not_exists to handle the case where the map doesn't exist yet.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param mapName The name of the map attribute
```

```
     * @param mapKey The key within the map to update
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateMapAttributeSafe(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String mapName,
        String mapKey,
        AttributeValue value) {

        // Create an empty map to use if the map doesn't exist
        Map<String, AttributeValue> emptyMap = new HashMap<>();
        AttributeValue emptyMapValue =
AttributeValue.builder().m(emptyMap).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #mapName = if_not_exists(#mapName, :emptyMap),
#mapName.#mapKey = :value")
            .expressionAttributeNames(Map.of(
                "#mapName", mapName,
                "#mapKey", mapKey))
            .expressionAttributeValues(Map.of(
                ":value",
                value,
                ":emptyMap",
                AttributeValue.builder().m(new HashMap<>()).build()))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Adds an attribute to a nested map.
     *
     * <p>This method demonstrates how to update a nested attribute without
     * overwriting the entire map.
```

```
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param path The path to the nested attribute as a list
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse addToNestedMap(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        List<String> path,
        AttributeValue value) {

        // Create expression attribute names for each part of the path
        Map<String, String> expressionAttributeNames = new HashMap<>();
        for (int i = 0; i < path.size(); i++) {
            expressionAttributeNames.put("#attr" + i, path.get(i));
        }

        // Build the attribute path using the expression attribute names
        StringBuilder attributePathExpression = new StringBuilder();
        for (int i = 0; i < path.size(); i++) {
            if (i > 0) {
                attributePathExpression.append(".");
            }
            attributePathExpression.append("#attr").append(i);
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET " + attributePathExpression.toString() + "
= :value")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
```

```
    }

    /**
     * Removes an attribute from a map.
     *
     * <p>This method demonstrates how to remove a specific attribute from a map.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param mapName The name of the map attribute
     * @param mapKey The key within the map to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse removeMapAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String mapName,
        String mapKey) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("REMOVE #mapName.#mapKey")
            .expressionAttributeNames(Map.of(
                "#mapName", mapName,
                "#mapKey", mapKey))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Creates a map with multiple attributes in a single operation.
     *
     * <p>This method demonstrates how to create a map with multiple attributes
     * in a single update operation.
     *
     * @param dynamoDbClient The DynamoDB client
```

```
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param mapName The name of the map attribute
     * @param attributes The attributes to set in the map
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse createMapWithAttributes(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String mapName,
        Map<String, AttributeValue> attributes) {

        // Create a map value from the attributes
        AttributeValue mapValue = AttributeValue.builder().m(attributes).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #mapName = :mapValue")
            .expressionAttributeNames(Map.of("#mapName", mapName))
            .expressionAttributeValues(Map.of(":mapValue", mapValue))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Gets the current value of a map attribute.
     *
     * <p>Helper method to retrieve the current value of a map attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param mapName The name of the map attribute
     * @return The map attribute value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, AttributeValue> getMapAttribute(
```

```
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String mapName) {

        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(mapName)
            .build();

        try {
            // Perform the get operation
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the map attribute if it exists, otherwise null
            if (response.item() != null && response.item().containsKey(mapName))
{
                return response.item().get(mapName).m();
            }

            return null;
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get map attribute: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

Ejemplo de uso de operaciones de asignación con AWS SDK for Java 2.x.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating map operations in DynamoDB");

        try {
            // Example 1: Create a map with multiple attributes
```

```
            System.out.println("\nExample 1: Creating a map with multiple
attributes");
            Map<String, AttributeValue> productDetails = new HashMap<>();
            productDetails.put("Color",
AttributeValue.builder().s("Red").build());
            productDetails.put("Weight",
AttributeValue.builder().n("2.5").build());
            productDetails.put(
                "Dimensions", AttributeValue.builder().s("10x20x5").build());

            UpdateItemResponse createResponse =
                createMapWithAttributes(dynamoDbClient, tableName, key,
"Details", productDetails);

            System.out.println("Created map attribute: " +
createResponse.attributes());

            // Example 2: Update a specific attribute in a map
            System.out.println("\nExample 2: Updating a specific attribute in a
map");
            UpdateItemResponse updateResponse = updateMapAttributeSafe(
                dynamoDbClient,
                tableName,
                key,
                "Details",
                "Color",
                AttributeValue.builder().s("Blue").build());

            System.out.println("Updated map attribute: " +
updateResponse.attributes());

            // Example 3: Add an attribute to a nested map
            System.out.println("\nExample 3: Adding an attribute to a nested
map");
            UpdateItemResponse nestedResponse = addToNestedMap(
                dynamoDbClient,
                tableName,
                key,
                List.of("Specifications", "Technical", "Resolution"),
                AttributeValue.builder().s("1920x1080").build());

            System.out.println("Added to nested map: " +
nestedResponse.attributes());
```

```
            // Example 4: Remove an attribute from a map
            System.out.println("\nExample 4: Removing an attribute from a map");
            UpdateItemResponse removeResponse =
                removeMapAttribute(dynamoDbClient, tableName, key, "Details",
"Dimensions");

            System.out.println("Updated map after removal: " +
removeResponse.attributes());

            // Example 5: Get the current value of a map attribute
            System.out.println("\nExample 5: Getting the current value of a map
attribute");
            Map<String, AttributeValue> currentMap =
getMapAttribute(dynamoDbClient, tableName, key, "Details");

            if (currentMap != null) {
                System.out.println("Current map attribute:");
                for (Map.Entry<String, AttributeValue> entry :
currentMap.entrySet()) {
                    System.out.println("  " + entry.getKey() + ": " +
entry.getValue());
                }
            } else {
                System.out.println("Map attribute not found");
            }

            // Explain map operations
            System.out.println("\nKey points about DynamoDB map operations:");
            System.out.println("1. Maps are unordered collections of name-value
pairs");
            System.out.println("2. Use dot notation (map.key) to access or update
specific attributes");
            System.out.println("3. You can update individual attributes without
overwriting the entire map");
            System.out.println("4. Maps can be nested to create complex data
structures");
            System.out.println("5. Use REMOVE to delete attributes from a map");
            System.out.println("6. You can create a map with multiple attributes
in a single operation");
            System.out.println("7. Map keys are case-sensitive");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
```

```
        }
    }
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Demuestre las operaciones de asignación con AWS SDK para JavaScript.

```
/**
 * Example of updating map attributes in DynamoDB.
 *
 * This module demonstrates how to update map attributes that may not exist,
 * how to update nested attributes, and how to handle various map update
 scenarios.
 */

const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update a map attribute safely, handling the case where the map might not
 exist.
 *
 * This function demonstrates using the if_not_exists function to safely update
 * a map attribute that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
```

```
 */
async function updateMapAttributeSafe(
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return response;
  } catch (error) {
    // If the error is because the map doesn't exist, create it
    if (error.name === "ValidationException" &&
        error.message.includes("The document path provided in the update
 expression is invalid")) {

      // Create the map with the specified key-value pair
      const createParams = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${mapName} = :map`,
        ExpressionAttributeValues: {
          ":map": { [mapKey]: value }
        },
        ReturnValues: "UPDATED_NEW"
      };
```

```
      return await docClient.send(new UpdateCommand(createParams));
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Update a map attribute using the if_not_exists function.
 *
 * This function demonstrates a more elegant approach using if_not_exists
 * to handle the case where the map doesn't exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMapAttributeWithIfNotExists(
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${mapName} = if_not_exists(${mapName}, :emptyMap),
 ${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
      ":emptyMap": {},
      ":value": value
```

```
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Add a value to a deeply nested map, creating parent maps if they don't exist.
 *
 * This function demonstrates how to update a deeply nested attribute,
 * creating any parent maps that don't exist along the way.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string[]} path - The path to the nested attribute as an array of keys
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function addToNestedMap(
  config,
  tableName,
  key,
  path,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = "SET";
  const expressionAttributeValues = {};

  // For each level in the path, create a map if it doesn't exist
  for (let i = 0; i < path.length; i++) {
    const currentPath = path.slice(0, i + 1).join(".");
    const parentPath = i > 0 ? path.slice(0, i).join(".") : null;

    if (parentPath) {
```

```
      updateExpression += ` ${parentPath} =
 if_not_exists(${parentPath}, :emptyMap${i}),`;
      expressionAttributeValues[`:emptyMap${i}`] = {};
    }
  }

  // Set the final value
  const fullPath = path.join(".");
  updateExpression += ` ${fullPath} = :value`;
  expressionAttributeValues[":value"] = value;

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update multiple fields in a map attribute in a single operation.
 *
 * This function demonstrates how to update multiple fields in a map
 * in a single DynamoDB operation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {Object} updates - Object containing key-value pairs to update
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMultipleMapFields(
  config,
  tableName,
  key,
  mapName,
```

```
  updates
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = `SET ${mapName} = if_not_exists(${mapName}, :emptyMap)`;
  const expressionAttributeValues = {
    ":emptyMap": {}
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([field, value], index) => {
    updateExpression += `, ${mapName}.${field} = :val${index}`;
    expressionAttributeValues[`:val${index}`] = value;
  });

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
```

```
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}

/**
 * Example of how to use the map attribute update functions.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Users";
  const key = { UserId: "U12345" };

  console.log("Demonstrating different approaches to update map attributes in
 DynamoDB");

  try {
    // Example 1: Update a map attribute that might not exist (two-step approach)
    console.log("\nExample 1: Updating a map attribute that might not exist (two-
step approach)");
    const response1 = await updateMapAttributeSafe(
      config,
      tableName,
      key,
      "Preferences",
      "Theme",
      "Dark"
```

```
    );

    console.log("Updated preferences:", response1.Attributes);

    // Example 2: Update a map attribute using if_not_exists (elegant approach)
    console.log("\nExample 2: Updating a map attribute using if_not_exists
(elegant approach)");
    const response2 = await updateMapAttributeWithIfNotExists(
      config,
      tableName,
      key,
      "Settings",
      "NotificationsEnabled",
      true
    );

    console.log("Updated settings:", response2.Attributes);

    // Example 3: Update a deeply nested attribute
    console.log("\nExample 3: Updating a deeply nested attribute");
    const response3 = await addToNestedMap(
      config,
      tableName,
      key,
      ["Profile", "Address", "City"],
      "Seattle"
    );

    console.log("Updated nested attribute:", response3.Attributes);

    // Example 4: Update multiple fields in a map
    console.log("\nExample 4: Updating multiple fields in a map");
    const response4 = await updateMultipleMapFields(
      config,
      tableName,
      key,
      "ContactInfo",
      {
        Email: "user@example.com",
        Phone: "555-123-4567",
        PreferredContact: "Email"
      }
    );
```

```
    console.log("Updated multiple fields:", response4.Attributes);

    // Get the final state of the item
    console.log("\nFinal state of the item:");
    const item = await getItem(config, tableName, key);
    console.log(JSON.stringify(item, null, 2));

    // Explain the benefits of different approaches
    console.log("\nKey points about updating map attributes:");
    console.log("1. Use if_not_exists to handle maps that might not exist");
    console.log("2. Multiple updates can be combined in a single operation");
    console.log("3. Deeply nested attributes require creating parent maps");
    console.log("4. DynamoDB expressions are atomic - the entire update succeeds
 or fails");
    console.log("5. Using a single operation is more efficient than multiple
 separate updates");

  } catch (error) {
    console.error("Error:", error);
  }
}

// Export the functions
module.exports = {
  updateMapAttributeSafe,
  updateMapAttributeWithIfNotExists,
  addToNestedMap,
  updateMultipleMapFields,
  getItem,
  exampleUsage
};

// Run the example if this file is executed directly
if (require.main === module) {
  exampleUsage();
}
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Demuestre las operaciones de asignación con AWS SDK for Python (Boto3).

```
"""
Example of updating map attributes in DynamoDB.

This module demonstrates how to update map attributes in DynamoDB, including
handling cases where the map attribute might not exist yet.
"""


import boto3
from typing import Any, Dict, Optional



def update_map_attribute_safe(
    table_name: str, key: Dict[str, Any], map_name: str, map_key: str, value: Any
) -> Dict[str, Any]:
    """
    Update a specific key in a map attribute, creating the map if it doesn't
 exist.

    This function demonstrates how to safely update a key within a map attribute,
    even if the map doesn't exist yet in the item.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        map_name (str): The name of the map attribute.
        map_key (str): The key within the map to update.
        value (Any): The value to set for the map key.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Use SET with attribute_not_exists to safely update the map
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #map.#key = :value",
        ExpressionAttributeNames={"#map": map_name, "#key": map_key},
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def add_to_nested_map(
    table_name: str, key: Dict[str, Any], path: str, value: Any
) -> Dict[str, Any]:
    """
    Add or update a value in a deeply nested map structure.

    This function demonstrates how to update a value at a specific path in a
    nested map structure, creating any intermediate maps as needed.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        path (str): The path to the nested attribute (e.g.,
 "user.preferences.theme").
        value (Any): The value to set at the specified path.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Split the path into components
    path_parts = path.split(".")

    # Build the update expression and attribute names
    update_expression = "SET "
    expression_attribute_names = {}
```

```
    # Build the path expression
    path_expression = ""
    for i, part in enumerate(path_parts):
        name_placeholder = f"#attr{i}"
        expression_attribute_names[name_placeholder] = part

        if i == 0:
            path_expression = name_placeholder
        else:
            path_expression += f".{name_placeholder}"

    # Complete the update expression
    update_expression += f"{path_expression} = :value"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def update_map_with_if_not_exists(
    table_name: str,
    key: Dict[str, Any],
    map_name: str,
    map_key: str,
    value: Any,
    default_map: Optional[Dict[str, Any]] = None,
) -> Dict[str, Any]:
    """
    Update a key in a map, creating the map with default values if it doesn't
 exist.

    This function demonstrates how to use if_not_exists to initialize a map with
    default values if it doesn't exist yet, and then update a specific key.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
```

```
        map_name (str): The name of the map attribute.
        map_key (str): The key within the map to update.
        value (Any): The value to set for the map key.
        default_map (Optional[Dict[str, Any]]): Default map values if the map
 doesn't exist.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Set default map if not provided
    if default_map is None:
        default_map = {}

    # Create a map with the new key-value pair
    updated_map = default_map.copy()
    updated_map[map_key] = value

    # Use if_not_exists to initialize the map if it doesn't exist
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #map = if_not_exists(#map, :default_map)",
        ExpressionAttributeNames={"#map": map_name},
        ExpressionAttributeValues={":default_map": updated_map},
        ReturnValues="UPDATED_NEW",
    )

    return response


def merge_into_map(
    table_name: str, key: Dict[str, Any], map_name: str, values_to_merge:
 Dict[str, Any]
) -> Dict[str, Any]:
    """
    Merge multiple key-value pairs into a map attribute.

    This function demonstrates how to update multiple keys in a map attribute
    in a single operation, without overwriting the entire map.
```

```
    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        map_name (str): The name of the map attribute.
        values_to_merge (Dict[str, Any]): Key-value pairs to merge into the map.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the update expression for each key-value pair
    update_expression = "SET "
    expression_attribute_names = {"#map": map_name}
    expression_attribute_values = {}

    # Add each key-value pair to the update expression
    for i, (k, v) in enumerate(values_to_merge.items()):
        key_placeholder = f"#key{i}"
        value_placeholder = f":value{i}"

        expression_attribute_names[key_placeholder] = k
        expression_attribute_values[value_placeholder] = v

        if i > 0:
            update_expression += ", "
        update_expression += f"#map.{key_placeholder} = {value_placeholder}"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues=expression_attribute_values,
        ReturnValues="UPDATED_NEW",
    )

    return response
```

```
def example_usage():
    """Example of how to use the map attribute update functions."""
    # Example parameters
    table_name = "UserProfiles"
    key = {"UserId": "user123"}

    print("Example 1: Updating a specific key in a map attribute")
    try:
        response = update_map_attribute_safe(
            table_name=table_name, key=key, map_name="Preferences",
 map_key="Theme", value="Dark"
        )
        print(f"Map attribute updated successfully: {response.get('Attributes',
 {})}")
    except Exception as e:
        print(f"Error updating map attribute: {e}")

    print("\nExample 2: Adding a value to a deeply nested map")
    try:
        response = add_to_nested_map(
            table_name=table_name, key=key, path="Settings.Notifications.Email",
 value=True
        )
        print(f"Nested map updated successfully: {response.get('Attributes',
 {})}")
    except Exception as e:
        print(f"Error updating nested map: {e}")

    print("\nExample 3: Initializing a map with default values if it doesn't
 exist")
    try:
        default_map = {"Language": "English", "Currency": "USD", "TimeZone":
 "UTC"}

        response = update_map_with_if_not_exists(
            table_name=table_name,
            key={"UserId": "newuser456"},
            map_name="Preferences",
            map_key="Theme",
            value="Light",
            default_map=default_map,
        )
        print(f"Map initialized with defaults: {response.get('Attributes', {})}")
```

```
    except Exception as e:
        print(f"Error initializing map: {e}")

    print("\nExample 4: Merging multiple values into a map")
    try:
        values_to_merge = {
            "NotificationsEnabled": True,
            "EmailFrequency": "Daily",
            "PushNotifications": False,
        }

        response = merge_into_map(
            table_name=table_name,
            key=key,
            map_name="NotificationSettings",
            values_to_merge=values_to_merge,
        )
        print(f"Multiple values merged into map: {response.get('Attributes',
 {})}")
    except Exception as e:
        print(f"Error merging values into map: {e}")

    print("\nBest practices for working with map attributes in DynamoDB:")
    print("1. Use dot notation to access and update nested attributes")
    print("2. Use ExpressionAttributeNames to handle reserved words and special
 characters")
    print("3. Use if_not_exists() to handle cases where attributes might not
 exist")
    print("4. Update specific map keys rather than overwriting the entire map")
    print("5. Use a single update operation to modify multiple map keys for
 better performance")
    print("6. Consider your data model carefully to minimize the need for deeply
 nested attributes")



if __name__ == "__main__":
    example_usage()
```

- Para obtener información la API, consulte UpdateItem en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Realización de operaciones de conjunto en DynamoDB con un AWS SDK

En los siguientes ejemplos de código se muestra cómo realizar operaciones de conjunto en DynamoDB.

- Agregue elementos a un atributo de conjunto.
- Elimine elementos de un atributo de conjunto.
- Utilice las operaciones ADD y DELETE con conjuntos.

Java

SDK para Java 2.x

Demuestre las operaciones de conjunto con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

    /**
     * Adds values to a string set attribute.
     *
     * <p>This method demonstrates how to use the ADD operation to add values
     * to a string set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
```

```
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param valuesToAdd The values to add to the set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse addToStringSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<String> valuesToAdd) {

        // Create a string set value from the values to add
        AttributeValue setValue =
AttributeValue.builder().ss(valuesToAdd).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("ADD #setAttr :valuesToAdd")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":valuesToAdd", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Adds values to a number set attribute.
     *
     * <p>This method demonstrates how to use the ADD operation to add values
     * to a number set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param valuesToAdd The values to add to the set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
```

```
     */
    public static UpdateItemResponse addToNumberSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<Number> valuesToAdd) {

        // Convert numbers to strings for DynamoDB
        Set<String> stringValues = new HashSet<>();
        for (Number value : valuesToAdd) {
            stringValues.add(value.toString());
        }

        // Create a number set value from the values to add
        AttributeValue setValue =
AttributeValue.builder().ns(stringValues).build();

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("ADD #setAttr :valuesToAdd")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":valuesToAdd", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Removes values from a set attribute.
     *
     * <p>This method demonstrates how to use the DELETE operation to remove
values
     * from a set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param valuesToRemove The values to remove from the set
```

```
     * @param isNumberSet Whether the set is a number set (true) or string set
(false)
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse removeFromSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<?> valuesToRemove,
        boolean isNumberSet) {

        AttributeValue setValue;

        if (isNumberSet) {
            // Convert numbers to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : valuesToRemove) {
                if (value instanceof Number) {
                    stringValues.add(value.toString());
                } else {
                    throw new IllegalArgumentException("Values must be numbers
for a number set");
                }
            }

            setValue = AttributeValue.builder().ns(stringValues).build();
        } else {
            // Convert objects to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : valuesToRemove) {
                stringValues.add(value.toString());
            }

            setValue = AttributeValue.builder().ss(stringValues).build();
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("DELETE #setAttr :valuesToRemove")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
```

```
            .expressionAttributeValues(Map.of(":valuesToRemove", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Checks if a value exists in a set attribute.
     *
     * <p>This method demonstrates how to use the contains function to check
     * if a value exists in a set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to check
     * @param setAttributeName The name of the set attribute
     * @param valueToCheck The value to check for
     * @return Map containing the result of the check
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, Object> checkIfValueInSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        String valueToCheck) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Define the update parameters with a condition expression
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #tempAttr = :tempVal")
                .conditionExpression("contains(#setAttr, :valueToCheck)")
                .expressionAttributeNames(Map.of("#setAttr", setAttributeName,
"#tempAttr", "TempAttribute"))
                .expressionAttributeValues(Map.of(
                    ":valueToCheck",
AttributeValue.builder().s(valueToCheck).build(),
                    ":tempVal", AttributeValue.builder().s("TempValue").build()))
```

```
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            // Attempt the update operation
            dynamoDbClient.updateItem(request);

            // If we get here, the condition was met
            result.put("exists", true);
            result.put("message", "Value '" + valueToCheck + "' exists in the
set");

            // Clean up the temporary attribute
            UpdateItemRequest cleanupRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("REMOVE #tempAttr")
                .expressionAttributeNames(Map.of("#tempAttr", "TempAttribute"))
                .build();

            dynamoDbClient.updateItem(cleanupRequest);

        } catch (DynamoDbException e) {
            if (e.getMessage().contains("ConditionalCheckFailed")) {
                // The condition was not met
                result.put("exists", false);
                result.put("message", "Value '" + valueToCheck + "' does not
exist in the set");
            } else {
                // Some other error occurred
                result.put("exists", false);
                result.put("message", "Error checking set: " + e.getMessage());
                result.put("error", e.getClass().getSimpleName());
            }
        }

        return result;
    }

    /**
     * Creates a set with multiple values in a single operation.
     *
     * <p>This method demonstrates how to create a set with multiple values
     * in a single update operation.
     *
```

```
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param setAttributeName The name of the set attribute
     * @param setValues The values to include in the set
     * @param isNumberSet Whether to create a number set (true) or string set
(false)
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse createSetWithValues(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String setAttributeName,
        Set<?> setValues,
        boolean isNumberSet) {

        AttributeValue setValue;

        if (isNumberSet) {
            // Convert numbers to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : setValues) {
                if (value instanceof Number) {
                    stringValues.add(value.toString());
                } else {
                    throw new IllegalArgumentException("Values must be numbers
for a number set");
                }
            }

            setValue = AttributeValue.builder().ns(stringValues).build();
        } else {
            // Convert objects to strings for DynamoDB
            Set<String> stringValues = new HashSet<>();
            for (Object value : setValues) {
                stringValues.add(value.toString());
            }

            setValue = AttributeValue.builder().ss(stringValues).build();
        }

        // Define the update parameters
```

```
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #setAttr = :setValue")
            .expressionAttributeNames(Map.of("#setAttr", setAttributeName))
            .expressionAttributeValues(Map.of(":setValue", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Gets the current value of a set attribute.
     *
     * <p>Helper method to retrieve the current value of a set attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param setAttributeName The name of the set attribute
     * @return The set attribute value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static AttributeValue getSetAttribute(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String setAttributeName) {

        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(setAttributeName)
            .build();

        try {
            // Perform the get operation
            GetItemResponse response = dynamoDbClient.getItem(request);

            // Return the set attribute if it exists, otherwise null
            if (response.item() != null &&
response.item().containsKey(setAttributeName)) {
                return response.item().get(setAttributeName);
```

```
            }

            return null;
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to get set attribute: " + e.getMessage())
                .cause(e)
                .build();
        }
    }
```

Ejemplo de uso de operaciones de conjunto con AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating set operations in DynamoDB");

        try {
            // Example 1: Create a string set with multiple values
            System.out.println("\nExample 1: Creating a string set with multiple
values");
            Set<String> tags = new HashSet<>();
            tags.add("Electronics");
            tags.add("Gadget");
            tags.add("Smartphone");

            UpdateItemResponse createResponse = createSetWithValues(
                dynamoDbClient, tableName, key, "Tags", tags, false // Not a
number set
                );

            System.out.println("Created set attribute: " +
createResponse.attributes());

            // Example 2: Add values to a string set
            System.out.println("\nExample 2: Adding values to a string set");
            Set<String> additionalTags = new HashSet<>();
            additionalTags.add("Mobile");
```

```
            additionalTags.add("Wireless");

            UpdateItemResponse addResponse = addToStringSet(dynamoDbClient,
tableName, key, "Tags", additionalTags);

            System.out.println("Updated set attribute: " +
addResponse.attributes());

            // Example 3: Create a number set with multiple values
            System.out.println("\nExample 3: Creating a number set with multiple
values");
            Set<Number> ratings = new HashSet<>();
            ratings.add(4);
            ratings.add(5);
            ratings.add(4.5);

            UpdateItemResponse createNumberSetResponse = createSetWithValues(
                dynamoDbClient, tableName, key, "Ratings", ratings, true // Is a
number set
                );

            System.out.println("Created number set attribute: " +
createNumberSetResponse.attributes());

            // Example 4: Add values to a number set
            System.out.println("\nExample 4: Adding values to a number set");
            Set<Number> additionalRatings = new HashSet<>();
            additionalRatings.add(3.5);
            additionalRatings.add(4.2);

            UpdateItemResponse addNumberResponse =
                addToNumberSet(dynamoDbClient, tableName, key, "Ratings",
additionalRatings);

            System.out.println("Updated number set attribute: " +
addNumberResponse.attributes());

            // Example 5: Remove values from a set
            System.out.println("\nExample 5: Removing values from a set");
            Set<String> tagsToRemove = new HashSet<>();
            tagsToRemove.add("Gadget");

            UpdateItemResponse removeResponse = removeFromSet(
```

```
                dynamoDbClient, tableName, key, "Tags", tagsToRemove, false // 
Not a number set
                );

            System.out.println("Updated set after removal: " + 
removeResponse.attributes());

            // Example 6: Check if a value exists in a set
            System.out.println("\nExample 6: Checking if a value exists in a 
set");
            Map<String, Object> checkResult = checkIfValueInSet(dynamoDbClient, 
tableName, key, "Tags", "Electronics");

            System.out.println("Check result: " + checkResult.get("message"));

            // Example 7: Get the current value of a set attribute
            System.out.println("\nExample 7: Getting the current value of a set 
attribute");
            AttributeValue currentStringSet = getSetAttribute(dynamoDbClient, 
tableName, key, "Tags");

            if (currentStringSet != null && currentStringSet.ss() != null) {
                System.out.println("Current string set values: " + 
currentStringSet.ss());
            } else {
                System.out.println("String set attribute not found");
            }

            AttributeValue currentNumberSet = getSetAttribute(dynamoDbClient, 
tableName, key, "Ratings");

            if (currentNumberSet != null && currentNumberSet.ns() != null) {
                System.out.println("Current number set values: " + 
currentNumberSet.ns());
            } else {
                System.out.println("Number set attribute not found");
            }

            // Explain set operations
            System.out.println("\nKey points about DynamoDB set operations:");
            System.out.println(
                "1. DynamoDB supports three set types: string sets (SS), number 
sets (NS), and binary sets (BS)");
```

```
            System.out.println("2. Sets can only contain elements of the same
 type");
            System.out.println("3. Use ADD to add elements to a set");
            System.out.println("4. Use DELETE to remove elements from a set");
            System.out.println("5. Sets automatically remove duplicate values");
            System.out.println("6. Sets are unordered collections");
            System.out.println("7. Use the contains function to check if a value
 exists in a set");
            System.out.println("8. You can create a set with multiple values in a
 single operation");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Demuestre las operaciones de conjunto con AWS SDK para JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Add elements to a set attribute.
 *
 * This function demonstrates using the ADD operation to add elements to a set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
```

```
 * @param {Array} values - The values to add to the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function addToSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using ADD
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `ADD ${setName} :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}
```

```
/**
 * Remove elements from a set attribute.
 *
 * This function demonstrates using the DELETE operation to remove elements from
 a set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The values to remove from the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeFromSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using DELETE
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `DELETE ${setName} :values`,
```

```
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Create a new set attribute with initial values.
 *
 * This function demonstrates using the SET operation to create a new set
 attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The initial values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
```

```
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using SET
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${setName} = :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Replace an entire set attribute with a new set of values.
 *
 * This function demonstrates using the SET operation to replace an entire set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The new values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function replaceSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
```

```
  // This is the same as createSet, but included for clarity of intent
  return await createSet(config, tableName, key, setName, values, setType);
}

/**
 * Remove the last element from a set and handle the empty set case.
 *
 * This function demonstrates what happens when you delete the last element of a
 set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @returns {Promise<Object>} - The result of the operation
 */
async function removeLastElementFromSet(
  config,
  tableName,
  key,
  setName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // First, get the current item to check the set
  const currentItem = await getItem(config, tableName, key);

  // Check if the set exists and has elements
  if (!currentItem || !currentItem[setName] || currentItem[setName].size === 0) {
    return {
      success: false,
      message: "Set doesn't exist or is already empty",
      item: currentItem
    };
  }

  // Get the set values
  const setValues = Array.from(currentItem[setName]);

  // If there's only one element left, remove the attribute entirely
  if (setValues.length === 1) {
    // Define the update parameters to remove the attribute
```

```
    const params = {
      TableName: tableName,
      Key: key,
      UpdateExpression: `REMOVE ${setName}`,
      ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Last element removed, attribute has been deleted",
      removedValue: setValues[0]
    };
  } else {
    // Otherwise, remove just the last element
    // Create a set with just the last element
    const lastElement = setValues[setValues.length - 1];
    const setType = typeof lastElement === 'number' ? 'number' : 'string';

    // Remove the last element
    const response = await removeFromSet(
      config,
      tableName,
      key,
      setName,
      [lastElement],
      setType
    );

    return {
      success: true,
      message: "Last element removed, set still contains elements",
      removedValue: lastElement,
      remainingSet: response.Attributes[setName]
    };
  }
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
```

```
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Ejemplo de uso de operaciones de conjunto con AWS SDK para JavaScript.

```
/**
 * Example of how to work with sets in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Users";
  const key = { UserId: "U12345" };

  console.log("Demonstrating set operations in DynamoDB");

  try {
```

```
    // Example 1: Create a string set
    console.log("\nExample 1: Creating a string set");
    const response1 = await createSet(
      config,
      tableName,
      key,
      "Interests",
      ["Reading", "Hiking", "Cooking"],
      "string"
    );

    console.log("Created set:", response1.Attributes);

    // Example 2: Add elements to a set
    console.log("\nExample 2: Adding elements to a set");
    const response2 = await addToSet(
      config,
      tableName,
      key,
      "Interests",
      ["Photography", "Travel"],
      "string"
    );

    console.log("Updated set after adding elements:", response2.Attributes);

    // Example 3: Remove elements from a set
    console.log("\nExample 3: Removing elements from a set");
    const response3 = await removeFromSet(
      config,
      tableName,
      key,
      "Interests",
      ["Cooking"],
      "string"
    );

    console.log("Updated set after removing elements:", response3.Attributes);

    // Example 4: Create a number set
    console.log("\nExample 4: Creating a number set");
    const response4 = await createSet(
      config,
      tableName,
```

```
    key,
    "FavoriteNumbers",
    [7, 42, 99],
    "number"
  );

  console.log("Created number set:", response4.Attributes);

  // Example 5: Replace an entire set
  console.log("\nExample 5: Replacing an entire set");
  const response5 = await replaceSet(
    config,
    tableName,
    key,
    "Interests",
    ["Gaming", "Movies", "Music"],
    "string"
  );

  console.log("Replaced set:", response5.Attributes);

  // Example 6: Remove the last element from a set
  console.log("\nExample 6: Removing the last element from a set");

  // First, create a set with just one element
  await createSet(
    config,
    tableName,
    { UserId: "U67890" },
    "Tags",
    ["LastTag"],
    "string"
  );

  // Then, remove the last element
  const response6 = await removeLastElementFromSet(
    config,
    tableName,
    { UserId: "U67890" },
    "Tags"
  );

  console.log(response6.message);
  console.log("Removed value:", response6.removedValue);
```

```
    // Get the final state of the items
    console.log("\nFinal state of the items:");
    const item1 = await getItem(config, tableName, key);
    console.log("User U12345:", JSON.stringify(item1, null, 2));

    const item2 = await getItem(config, tableName, { UserId: "U67890" });
    console.log("User U67890:", JSON.stringify(item2, null, 2));

    // Explain set operations
    console.log("\nKey points about set operations in DynamoDB:");
    console.log("1. Use ADD to add elements to a set (duplicates are
 automatically removed)");
    console.log("2. Use DELETE to remove elements from a set");
    console.log("3. Use SET to create a new set or replace an existing one");
    console.log("4. DynamoDB supports three types of sets: string sets, number
 sets, and binary sets");
    console.log("5. When you delete the last element from a set, the attribute
 remains as an empty set");
    console.log("6. To remove an empty set, use the REMOVE operation");
    console.log("7. Sets automatically maintain unique values (no duplicates)");
    console.log("8. You cannot mix data types within a set");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Demuestre las operaciones de conjunto con AWS SDK for Python (Boto3).

```
import boto3
from typing import Any, Dict, List


def create_set_attribute(
```

```
    table_name: str,
    key: Dict[str, Any],
    set_name: str,
    set_values: List[Any],
    set_type: str = "string",
) -> Dict[str, Any]:
    """
    Create a new set attribute or add elements to an existing set.

    This function demonstrates how to use the ADD operation to create a new set
    or add elements to an existing set.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        set_values (List[Any]): The values to add to the set.
        set_type (str, optional): The type of set to create: "string", "number",
 or "binary".
            Defaults to "string".

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Convert the list to a DynamoDB set based on the specified type
    if set_type == "string":
        dynamo_set = set(str(value) for value in set_values)
    elif set_type == "number":
        # We need to use actual float values for the DynamoDB API
        # but mypy expects strings in sets, so we need to use type: ignore
        dynamo_set = set(float(value) for value in set_values)  # type: ignore
    else:  # binary set is not directly supported in high-level API, handled
 differently
        raise ValueError("Binary sets are not supported in this example")

    # Use the ADD operation to create or update the set
    response = table.update_item(
        Key=key,
        UpdateExpression="ADD #set_attr :set_values",
```

```
        ExpressionAttributeNames={"#set_attr": set_name},
        ExpressionAttributeValues={":set_values": dynamo_set},
        ReturnValues="UPDATED_NEW",
    )

    return response


def add_to_set(
    table_name: str, key: Dict[str, Any], set_name: str, values_to_add: List[Any]
) -> Dict[str, Any]:
    """
    Add elements to an existing set attribute.

    This function demonstrates how to use the ADD operation to add elements to an
 existing set.
    If the set doesn't exist, it will be created.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        values_to_add (List[Any]): The values to add to the set.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Convert the list to a set (assuming string set for simplicity)
    dynamo_set = set(str(value) for value in values_to_add)

    # Use the ADD operation to add values to the set
    response = table.update_item(
        Key=key,
        UpdateExpression="ADD #set_attr :values_to_add",
        ExpressionAttributeNames={"#set_attr": set_name},
        ExpressionAttributeValues={":values_to_add": dynamo_set},
        ReturnValues="UPDATED_NEW",
    )
```

```
    return response


def remove_from_set(
    table_name: str, key: Dict[str, Any], set_name: str, values_to_remove:
 List[Any]
) -> Dict[str, Any]:
    """
    Remove elements from a set attribute.

    This function demonstrates how to use the DELETE operation to remove elements
 from a set.
    If the last element is removed, the attribute will be deleted entirely.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        values_to_remove (List[Any]): The values to remove from the set.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Convert the list to a set (assuming string set for simplicity)
    dynamo_set = set(str(value) for value in values_to_remove)

    # Use the DELETE operation to remove values from the set
    response = table.update_item(
        Key=key,
        UpdateExpression="DELETE #set_attr :values_to_remove",
        ExpressionAttributeNames={"#set_attr": set_name},
        ExpressionAttributeValues={":values_to_remove": dynamo_set},
        ReturnValues="UPDATED_NEW",
    )

    return response
```

```
def check_if_set_exists(table_name: str, key: Dict[str, Any], set_name: str) ->
 bool:
    """
    Check if a set attribute exists in an item.

    This function demonstrates how to check if a set attribute exists after
    potentially removing all elements from it.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to check.
        set_name (str): The name of the set attribute.

    Returns:
        bool: True if the set attribute exists, False otherwise.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Get the item
    response = table.get_item(
        Key=key, ProjectionExpression="#set_attr",
 ExpressionAttributeNames={"#set_attr": set_name}
    )

    # Check if the item exists and has the set attribute
    return "Item" in response and set_name in response["Item"]


def demonstrate_last_element_removal(
    table_name: str, key: Dict[str, Any], set_name: str
) -> Dict[str, Any]:
    """
    Demonstrate what happens when you remove the last element from a set.

    This function creates a set with a single element, then removes that element,
    showing that the attribute is completely removed when the last element is
 deleted.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
```

```
    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Step 1: Create a set with a single element
    create_response = create_set_attribute(
        table_name=table_name,
        key=key,
        set_name=set_name,
        set_values=["last_element"],
        set_type="string",
    )

    # Step 2: Check that the set exists
    exists_before = check_if_set_exists(table_name, key, set_name)

    # Step 3: Remove the last element
    delete_response = remove_from_set(
        table_name=table_name, key=key, set_name=set_name,
 values_to_remove=["last_element"]
    )

    # Step 4: Check if the set still exists
    exists_after = check_if_set_exists(table_name, key, set_name)

    # Return the results
    return {
        "create_response": create_response,
        "exists_before": exists_before,
        "delete_response": delete_response,
        "exists_after": exists_after,
    }


def work_with_number_set(
    table_name: str,
    key: Dict[str, Any],
    set_name: str,
    initial_values: List[float],
    values_to_add: List[float],
    values_to_remove: List[float],
) -> Dict[str, Any]:
    """
    Demonstrate working with a number set in DynamoDB.
```

```
    This function shows how to create and manipulate a set of numbers.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        set_name (str): The name of the set attribute.
        initial_values (List[float]): The initial values for the set.
        values_to_add (List[float]): Values to add to the set.
        values_to_remove (List[float]): Values to remove from the set.

    Returns:
        Dict[str, Any]: A dictionary containing the responses from each
operation.
    """
    # Step 1: Create the number set
    create_response = create_set_attribute(
        table_name=table_name,
        key=key,
        set_name=set_name,
        set_values=initial_values,
        set_type="number",
    )

    # Step 2: Add more numbers to the set
    add_response = add_to_set(
        table_name=table_name, key=key, set_name=set_name,
values_to_add=values_to_add
    )

    # Step 3: Remove some numbers from the set
    remove_response = remove_from_set(
        table_name=table_name, key=key, set_name=set_name,
values_to_remove=values_to_remove
    )

    # Step 4: Get the final state
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    get_response = table.get_item(
        Key=key,
        ProjectionExpression=f"#{set_name}",
        ExpressionAttributeNames={f"#{set_name}": set_name},
```

```
    )

    # Return all responses
    return {
        "create_response": create_response,
        "add_response": add_response,
        "remove_response": remove_response,
        "final_state": get_response.get("Item", {}),
    }
```

Ejemplo de uso de operaciones de conjunto con AWS SDK for Python (Boto3).

```
def example_usage():
    """Example of how to use the set operations functions."""
    # Example parameters
    table_name = "UserPreferences"
    key = {"UserId": "user123"}

    print("Example 1: Creating a string set attribute")
    try:
        response = create_set_attribute(
            table_name=table_name,
            key=key,
            set_name="FavoriteTags",
            set_values=["AWS", "DynamoDB", "NoSQL"],
            set_type="string",
        )
        print(f"Set attribute created successfully: {response.get('Attributes',
 {})}")
    except Exception as e:
        print(f"Error creating set attribute: {e}")

    print("\nExample 2: Adding elements to an existing set")
    try:
        response = add_to_set(
            table_name=table_name,
            key=key,
            set_name="FavoriteTags",
            values_to_add=["Database", "Serverless"],
        )
```

```
        print(f"Elements added to set successfully: {response.get('Attributes',
{})}")
    except Exception as e:
        print(f"Error adding to set: {e}")

    print("\nExample 3: Removing elements from a set")
    try:
        response = remove_from_set(
            table_name=table_name, key=key, set_name="FavoriteTags",
values_to_remove=["NoSQL"]
        )
        print(f"Elements removed from set successfully:
{response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error removing from set: {e}")

    print("\nExample 4: Demonstrating what happens when you remove the last
element from a set")
    try:
        results = demonstrate_last_element_removal(
            table_name=table_name, key={"UserId": "tempUser"},
set_name="SingleElementSet"
        )

        print(f"Set exists before removal: {results['exists_before']}")
        print(f"Set exists after removal: {results['exists_after']}")

        if not results["exists_after"]:
            print("The set attribute was completely removed when the last element
was deleted.")
        else:
            print("The set attribute still exists after removing the last
element.")
    except Exception as e:
        print(f"Error in last element removal demonstration: {e}")

    print("\nExample 5: Working with a number set")
    try:
        results = work_with_number_set(
            table_name=table_name,
            key={"UserId": "user456"},
            set_name="LuckyNumbers",
            initial_values=[7, 13, 42],
            values_to_add=[99, 100],
```

```
            values_to_remove=[13],
        )

        print(f"Initial number set: {results['create_response'].get('Attributes',
{})}")
        print(f"After adding numbers: {results['add_response'].get('Attributes',
{})}")
        print(f"After removing numbers:
{results['remove_response'].get('Attributes', {})}")
        print(f"Final state: {results['final_state']}")
    except Exception as e:
        print(f"Error working with number set: {e}")

    print("\nKey Points About DynamoDB Sets:")
    print("1. Sets can only contain elements of the same type (string, number, or
binary)")
    print("2. Sets automatically eliminate duplicate values")
    print("3. The ADD operation creates a set if it doesn't exist")
    print("4. The DELETE operation removes specified elements from a set")
    print("5. When the last element is removed from a set, the entire attribute
is deleted")
    print("6. Empty sets are not allowed in DynamoDB")
    print("7. Sets are unordered collections")
    print("8. The ADD operation is atomic for sets")
```

- Para obtener información la API, consulte UpdateItem en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Consultar una tabla de DynamoDB mediante lotes de instrucciones PartiQL y un SDK de AWS

En el siguiente ejemplo de código, se muestra cómo:

- Obtener un lote de elementos mediante la ejecución de varias instrucciones SELECT.

- Agregar un lote de elementos mediante la ejecución de varias instrucciones INSERT.
- Actualizar un lote de elementos con la ejecución de varias instrucciones UPDATE.
- Eliminar un lote de elementos con la ejecución de varias instrucciones DELETE.

.NET

SDK para .NET

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
// Before you run this example, download 'movies.json' from
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
GettingStarted.Js.02.html,
// and put it in the same folder as the example.

// Separator for the console display.
var SepBar = new string('-', 80);
const string tableName = "movie_table";
const string movieFileName = @"..\..\..\..\..\..\..\resources\sample_files
\movies.json";

DisplayInstructions();

// Create the table and wait for it to be active.
Console.WriteLine($"Creating the movie table: {tableName}");

var success = await DynamoDBMethods.CreateMovieTableAsync(tableName);
if (success)
{
    Console.WriteLine($"Successfully created table: {tableName}.");
}

WaitForEnter();

// Add movie information to the table from moviedata.json. See the
// instructions at the top of this file to download the JSON file.
```

```
Console.WriteLine($"Inserting movies into the new table. Please wait...");
success = await PartiQLBatchMethods.InsertMovies(tableName, movieFileName);
if (success)
{
    Console.WriteLine("Movies successfully added to the table.");
}
else
{
    Console.WriteLine("Movies could not be added to the table.");
}

WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var title1 = "Star Wars";
var year1 = 1977;
var title2 = "Wizard of Oz";
var year2 = 1939;

Console.WriteLine($"Updating two movies with producer information: {title1} and
 {title2}.");
success = await PartiQLBatchMethods.GetBatch(tableName, title1, title2, year1,
 year2);
if (success)
{
    Console.WriteLine($"Successfully retrieved {title1} and {title2}.");
}
else
{
    Console.WriteLine("Select statement failed.");
}

WaitForEnter();

// Update multiple movies by using the BatchExecute statement.
var producer1 = "LucasFilm";
var producer2 = "MGM";

Console.WriteLine($"Updating two movies with producer information: {title1} and
 {title2}.");
success = await PartiQLBatchMethods.UpdateBatch(tableName, producer1, title1,
 year1, producer2, title2, year2);
if (success)
{
```

```
    Console.WriteLine($"Successfully updated {title1} and {title2}.");
}
else
{
    Console.WriteLine("Update failed.");
}

WaitForEnter();

// Delete multiple movies by using the BatchExecute statement.
Console.WriteLine($"Now we will delete {title1} and {title2} from the table.");
success = await PartiQLBatchMethods.DeleteBatch(tableName, title1, year1, title2,
 year2);

if (success)
{
    Console.WriteLine($"Deleted {title1} and {title2}");
}
else
{
    Console.WriteLine($"could not delete {title1} or {title2}");
}

WaitForEnter();

// DNow that the PartiQL Batch scenario is complete, delete the movie table.
success = await DynamoDBMethods.DeleteTableAsync(tableName);

if (success)
{
    Console.WriteLine($"Successfully deleted {tableName}");
}
else
{
    Console.WriteLine($"Could not delete {tableName}");
}

/// <summary>
/// Displays the description of the application on the console.
/// </summary>
void DisplayInstructions()
{
    Console.Clear();
    Console.WriteLine();
```

```
    Console.Write(new string(' ', 24));
    Console.WriteLine("DynamoDB PartiQL Basics Example");
    Console.WriteLine(SepBar);
    Console.WriteLine("This demo application shows the basics of using Amazon
 DynamoDB with the AWS SDK for");
    Console.WriteLine(".NET version 3.7 and .NET 6.");
    Console.WriteLine(SepBar);
    Console.WriteLine("Creates a table by using the CreateTable method.");
    Console.WriteLine("Gets multiple movies by using a PartiQL SELECT
 statement.");
    Console.WriteLine("Updates multiple movies by using the ExecuteBatch
 method.");
    Console.WriteLine("Deletes multiple movies by using a PartiQL DELETE
 statement.");
    Console.WriteLine("Cleans up the resources created for the demo by deleting
 the table.");
    Console.WriteLine(SepBar);

    WaitForEnter();
}

/// <summary>
/// Simple method to wait for the <Enter> key to be pressed.
/// </summary>
void WaitForEnter()
{
    Console.WriteLine("\nPress <Enter> to continue.");
    Console.Write(SepBar);
    _ = Console.ReadLine();
}
```

```
        /// <summary>
        /// Gets movies from the movie table by
        /// using an Amazon DynamoDB PartiQL SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year1">The year of the first movie.</param>
        /// <param name="year2">The year of the second movie.</param>
        /// <returns>True if successful.</returns>
        public static async Task<bool> GetBatch(
            string tableName,
```

```
            string title1,
            string title2,
            int year1,
            int year2)
        {
            var getBatch = $"SELECT * FROM {tableName} WHERE title = ? AND year
= ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = getBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            if (response.Responses.Count > 0)
            {
                response.Responses.ForEach(r =>
                {
                    if (r.Item.Any())
                    {

Console.WriteLine($"{r.Item["title"]}\t{r.Item["year"]}");
```

```
                    }
                });
                return true;
            }
            else
            {
                Console.WriteLine($"Couldn't find either {title1} or {title2}.");
                return false;
            }

        }

        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table into which the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
```

```
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
```

```
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }

            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }

        /// <summary>
        /// Updates information for multiple movies.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// movies to be updated.</param>
        /// <param name="producer1">The producer name for the first movie
        /// to update.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year that the first movie was released.</
param>
        /// <param name="producer2">The producer name for the second
        /// movie to update.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year that the second movie was released.</
param>
        /// <returns>A Boolean value that indicates the success of the update.</
returns>
        public static async Task<bool> UpdateBatch(
            string tableName,
            string producer1,
```

```
            string title1,
            int year1,
            string producer2,
            string title2,
            int year2)
        {

            string updateBatch = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer1 },
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = producer2 },
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };

            var response = await Client.BatchExecuteStatementAsync(new
BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }

        /// <summary>
```

```
        /// Deletes multiple movies using a PartiQL BatchExecuteAsync
        /// statement.
        /// </summary>
        /// <param name="tableName">The name of the table containing the
        /// moves that will be deleted.</param>
        /// <param name="title1">The title of the first movie.</param>
        /// <param name="year1">The year the first movie was released.</param>
        /// <param name="title2">The title of the second movie.</param>
        /// <param name="year2">The year the second movie was released.</param>
        /// <returns>A Boolean value indicating the success of the operation.</
returns>
        public static async Task<bool> DeleteBatch(
            string tableName,
            string title1,
            int year1,
            string title2,
            int year2)
        {

            string updateBatch = $"DELETE FROM {tableName} WHERE title = ? AND
 year = ?";
            var statements = new List<BatchStatementRequest>
            {
                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title1 },
                        new AttributeValue { N = year1.ToString() },
                    },
                },

                new BatchStatementRequest
                {
                    Statement = updateBatch,
                    Parameters = new List<AttributeValue>
                    {
                        new AttributeValue { S = title2 },
                        new AttributeValue { N = year2.ToString() },
                    },
                }
            };
```

```
            var response = await Client.BatchExecuteStatementAsync(new
 BatchExecuteStatementRequest
            {
                Statements = statements,
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Para obtener información sobre la API, consulte BatchExecuteStatement en la referencia de la API de AWS SDK para .NET.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        //  1. Create a table. (CreateTable)
        if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

            AwsDoc::DynamoDB::partiqlBatchExecuteScenario(clientConfig);

            // 7. Delete the table. (DeleteTable)
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
        }

//! Scenario to modify and query a DynamoDB table using PartiQL batch statements.
/*!
  \sa partiqlBatchExecuteScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::DynamoDB::partiqlBatchExecuteScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
```

```
    // 2. Add multiple movies using "Insert" statements. (BatchExecuteStatement)
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    std::vector<Aws::String> titles;
    std::vector<float> ratings;
    std::vector<int> years;
    std::vector<Aws::String> plots;
    Aws::String doAgain = "n";
    do {
        Aws::String aTitle = askQuestion(
                "Enter the title of a movie you want to add to the table: ");
        titles.push_back(aTitle);
        int aYear = askQuestionForInt("What year was it released? ");
        years.push_back(aYear);
        float aRating = askQuestionForFloatRange(
                "On a scale of 1 - 10, how do you rate it? ",
                1, 10);
        ratings.push_back(aRating);
        Aws::String aPlot = askQuestion("Summarize the plot for me: ");
        plots.push_back(aPlot);

        doAgain = askQuestion(Aws::String("Would you like to add more movies? (y/
n) "));
    } while (doAgain == "y");

    std::cout << "Adding " << titles.size()
              << (titles.size() == 1 ? " movie " : " movies ")
              << "to the table using a batch \"INSERT\" statement." << std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                  << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                  << INFO_KEY << "': ?}";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
```

```
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));

            // Create attribute for the info map.
            Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute
= Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            ratingAttribute->SetN(ratings[i]);
            infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

            std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
                    ALLOCATION_TAG.c_str());
            plotAttribute->SetS(plots[i]);
            infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
            attributes.push_back(infoMapAttribute);
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add the movies: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    std::cout << "Retrieving the movie data with a batch \"SELECT\" statement."
              << std::endl;

    // 3. Get the data for multiple movies using "Select" statements.
(BatchExecuteStatement)
```

```
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (outcome.IsSuccess()) {
            const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

            for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

                printMovieInfo(item);
            }
        }
        else {
            std::cerr << "Failed to retrieve the movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
```

```
            return false;
        }
    }

    // 4. Update the data for multiple movies using "Update" statements.
(BatchExecuteStatement)

    for (size_t i = 0; i < titles.size(); ++i) {
        ratings[i] = askQuestionForFloatRange(
                Aws::String("\nLet's update your the movie, \"") + titles[i] +
                ".\nYou rated it  " + std::to_string(ratings[i])
                + ", what new rating would you give it? ", 1, 10);
    }

    std::cout << "Updating the movie with a batch \"UPDATE\" statement." <<
std::endl;

    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());

        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";


        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);

            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetN(ratings[i]));
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;
```

```
        request.SetStatements(statements);
        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    std::cout << "Retrieving the updated movie data with a batch \"SELECT\"
statement."
              << std::endl;

    // 5. Get the updated data for multiple movies using "Select" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }

        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
dynamoClient.BatchExecuteStatement(
                request);
        if (outcome.IsSuccess()) {
```

```
            const Aws::DynamoDB::Model::BatchExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::DynamoDB::Model::BatchStatementResponse>
&responses = result.GetResponses();

            for (const Aws::DynamoDB::Model::BatchStatementResponse &response:
responses) {
                const Aws::Map<Aws::String, Aws::DynamoDB::Model::AttributeValue>
&item = response.GetItem();

                printMovieInfo(item);
            }
        }
        else {
            std::cerr << "Failed to retrieve the movies information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    std::cout << "Deleting the movie data with a batch \"DELETE\" statement."
              << std::endl;

    // 6. Delete multiple movies using "Delete" statements.
(BatchExecuteStatement)
    {
        Aws::Vector<Aws::DynamoDB::Model::BatchStatementRequest> statements(
                titles.size());
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        std::string sql(sqlStream.str());

        for (size_t i = 0; i < statements.size(); ++i) {
            statements[i].SetStatement(sql);
            Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
            attributes.push_back(
                    Aws::DynamoDB::Model::AttributeValue().SetS(titles[i]));

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(years[i]));
            statements[i].SetParameters(attributes);
        }
```

```
        Aws::DynamoDB::Model::BatchExecuteStatementRequest request;

        request.SetStatements(statements);

        Aws::DynamoDB::Model::BatchExecuteStatementOutcome outcome =
 dynamoClient.BatchExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movies: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    return true;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;

    {
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
```

```
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
                std::cerr << "Failed to create table: "
                          << result.GetError().GetMessage();
                return false;
            }
        }
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
clientConfiguration)) {
```

```
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
```

```
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
        count++;
    }
    return false;
}
```

- Para obtener información sobre la API, consulte BatchExecuteStatement en la referencia de la API de AWS SDK para C++.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Ejecutar un escenario que crea una tabla y ejecuta lotes de consultas PartiQL.

```
import (
 "context"
 "fmt"
 "log"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunPartiQLBatchScenario shows you how to use the AWS SDK for Go
// to run batches of PartiQL statements to query a table that stores data about
 movies.
//
//   - Use batches of PartiQL statements to add, get, update, and delete data for
//     individual movies.
//
// This example creates an Amazon DynamoDB service client from the specified
 sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// This example creates and deletes a DynamoDB table to use during the scenario.
func RunPartiQLBatchScenario(ctx context.Context, sdkConfig aws.Config, tableName
 string) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
```

```
}()

log.Println(strings.Repeat("-", 88))
log.Println("Welcome to the Amazon DynamoDB PartiQL batch demo.")
log.Println(strings.Repeat("-", 88))

tableBasics := actions.TableBasics{
 DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
 TableName:      tableName,
}
runner := actions.PartiQLRunner{
 DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
 TableName:      tableName,
}

exists, err := tableBasics.TableExists(ctx)
if err != nil {
 panic(err)
}
if !exists {
 log.Printf("Creating table %v...\n", tableName)
 _, err = tableBasics.CreateMovieTable(ctx)
 if err != nil {
  panic(err)
 } else {
  log.Printf("Created table %v.\n", tableName)
 }
} else {
 log.Printf("Table %v already exists.\n", tableName)
}
log.Println(strings.Repeat("-", 88))

currentYear, _, _ := time.Now().Date()
customMovies := []actions.Movie{{
 Title: "House PartiQL",
 Year:  currentYear - 5,
 Info: map[string]interface{}{
  "plot":   "Wacky high jinks result from querying a mysterious database.",
  "rating": 8.5}}, {
 Title: "House PartiQL 2",
 Year:  currentYear - 3,
 Info: map[string]interface{}{
  "plot":   "Moderate high jinks result from querying another mysterious
database.",
```

```
  "rating": 6.5}}, {
 Title: "House PartiQL 3",
 Year:  currentYear - 1,
 Info: map[string]interface{}{
  "plot":   "Tepid high jinks result from querying yet another mysterious
database.",
  "rating": 2.5},
},
}

log.Printf("Inserting a batch of movies into table '%v'.\n", tableName)
err = runner.AddMovieBatch(ctx, customMovies)
if err == nil {
 log.Printf("Added %v movies to the table.\n", len(customMovies))
}
log.Println(strings.Repeat("-", 88))

log.Println("Getting data for a batch of movies.")
movies, err := runner.GetMovieBatch(ctx, customMovies)
if err == nil {
 for _, movie := range movies {
  log.Println(movie)
 }
}
log.Println(strings.Repeat("-", 88))

newRatings := []float64{7.7, 4.4, 1.1}
log.Println("Updating a batch of movies with new ratings.")
err = runner.UpdateMovieBatch(ctx, customMovies, newRatings)
if err == nil {
 log.Printf("Updated %v movies with new ratings.\n", len(customMovies))
}
log.Println(strings.Repeat("-", 88))

log.Println("Getting projected data from the table to verify our update.")
log.Println("Using a page size of 2 to demonstrate paging.")
projections, err := runner.GetAllMovies(ctx, 2)
if err == nil {
 log.Println("All movies:")
 for _, projection := range projections {
  log.Println(projection)
 }
}
log.Println(strings.Repeat("-", 88))
```

```
 log.Println("Deleting a batch of movies.")
 err = runner.DeleteMovieBatch(ctx, customMovies)
 if err == nil {
  log.Printf("Deleted %v movies.\n", len(customMovies))
 }

 err = tableBasics.DeleteTable(ctx)
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Defina una estructura Película para utilizar en este ejemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
```

```
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Crear una estructura y métodos que ejecuten instrucciones PartiQL.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
```

```
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies
 to the
// DynamoDB table.
func (runner PartiQLRunner) AddMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year, movie.Info})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(fmt.Sprintf(
    "INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't insert a batch of items with PartiQL. Here's why: %v\n",
 err)
 }
 return err
}



// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
 from
// the DynamoDB table by title and year.
```

```
func (runner PartiQLRunner) GetMovieBatch(ctx context.Context, movies []Movie)
 ([]Movie, error) {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 output, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 var outMovies []Movie
 if err != nil {
  log.Printf("Couldn't get a batch of items with PartiQL. Here's why: %v\n", err)
 } else {
  for _, response := range output.Responses {
   var movie Movie
   err = attributevalue.UnmarshalMap(response.Item, &movie)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    outMovies = append(outMovies, movie)
   }
  }
 }
 return outMovies, err
}



// GetAllMovies runs a PartiQL SELECT statement to get all movies from the
 DynamoDB table.
// pageSize is not typically required and is used to show how to paginate the
 results.
```

```
// The results are projected to return only the title and rating of each movie.
func (runner PartiQLRunner) GetAllMovies(ctx context.Context, pageSize int32)
 ([]map[string]interface{}, error) {
 var output []map[string]interface{}
 var response *dynamodb.ExecuteStatementOutput
 var err error
 var nextToken *string
 for moreData := true; moreData; {
  response, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
   Statement: aws.String(
    fmt.Sprintf("SELECT title, info.rating FROM \"%v\"", runner.TableName)),
   Limit:     aws.Int32(pageSize),
   NextToken: nextToken,
  })
  if err != nil {
   log.Printf("Couldn't get movies. Here's why: %v\n", err)
   moreData = false
  } else {
   var pageOutput []map[string]interface{}
   err = attributevalue.UnmarshalListOfMaps(response.Items, &pageOutput)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    log.Printf("Got a page of length %v.\n", len(response.Items))
    output = append(output, pageOutput...)
   }
   nextToken = response.NextToken
   moreData = nextToken != nil
  }
 }
 return output, err
}



// UpdateMovieBatch runs a batch of PartiQL UPDATE statements to update the
 rating of
// multiple movies that already exist in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovieBatch(ctx context.Context, movies []Movie,
 ratings []float64) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
```

```
  params, err := attributevalue.MarshalList([]interface{}{ratings[index],
 movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
 }
 return err
}



// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
 movies
// from the DynamoDB table.
func (runner PartiQLRunner) DeleteMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }
```

```
 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

- Para obtener información sobre la API, consulte BatchExecuteStatement en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
public class ScenarioPartiQLBatch {
    public static void main(String[] args) throws IOException {
        String tableName = "MoviesPartiQBatch";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        System.out.println("Creating an Amazon DynamoDB table named " + tableName
            + " with a key named year and a sort key named title.");
        createTable(ddb, tableName);

        System.out.println("Adding multiple records into the " + tableName
            + " table using a batch command.");
        putRecordBatch(ddb);
```

```
        // Update multiple movies by using the BatchExecute statement.
        String title1 = "Star Wars";
        int year1 = 1977;
        String title2 = "Wizard of Oz";
        int year2 = 1939;

        System.out.println("Query two movies.");
        getBatch(ddb, tableName, title1, title2, year1, year2);

        System.out.println("Updating multiple records using a batch command.");
        updateTableItemBatch(ddb);

        System.out.println("Deleting multiple records using a batch command.");
        deleteItemBatch(ddb);

        System.out.println("Deleting the Amazon DynamoDB table.");
        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static boolean getBatch(DynamoDbClient ddb, String tableName, String
title1, String title2, int year1, int year2) {
        String getBatch = "SELECT * FROM " + tableName + " WHERE title = ? AND
year = ?";

        List<BatchStatementRequest> statements = new ArrayList<>();
        statements.add(BatchStatementRequest.builder()
            .statement(getBatch)
            .parameters(AttributeValue.builder().s(title1).build(),
                AttributeValue.builder().n(String.valueOf(year1)).build())
            .build());
        statements.add(BatchStatementRequest.builder()
            .statement(getBatch)
            .parameters(AttributeValue.builder().s(title2).build(),
                AttributeValue.builder().n(String.valueOf(year2)).build())
            .build());

        BatchExecuteStatementRequest batchExecuteStatementRequest =
BatchExecuteStatementRequest.builder()
            .statements(statements)
            .build();

        try {
```

```
            BatchExecuteStatementResponse response =
ddb.batchExecuteStatement(batchExecuteStatementRequest);
            if (!response.responses().isEmpty()) {
                response.responses().forEach(r -> {
                    System.out.println(r.item().get("title") + "\\t" +
r.item().get("year"));
                });
                return true;
            } else {
                System.out.println("Couldn't find either " + title1 + " or " +
title2 + ".");
                return false;
            }
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            return false;
        }
    }

    public static void createTable(DynamoDbClient ddb, String tableName) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("year")
            .attributeType("N")
            .build());

        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
            .attributeName("title")
            .keyType(KeyType.RANGE) // Sort
            .build();
```

```
        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //  DynamoDB automatically
scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse = dbWaiter
                .waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void putRecordBatch(DynamoDbClient ddb) {
        String sqlStatement = "INSERT INTO MoviesPartiQBatch VALUE {'year':?,
'title' : ?, 'info' : ?}";
        try {
            // Create three movies to add to the Amazon DynamoDB table.
            // Set data for Movie 1.
            List<AttributeValue> parameters = new ArrayList<>();

            AttributeValue att1 = AttributeValue.builder()
                .n("1977")
                .build();
```

```
            AttributeValue att2 = AttributeValue.builder()
                .s("Star Wars")
                .build();

            AttributeValue att3 = AttributeValue.builder()
                .s("No Information")
                .build();

            parameters.add(att1);
            parameters.add(att2);
            parameters.add(att3);

            BatchStatementRequest statementRequestMovie1 =
BatchStatementRequest.builder()
                .statement(sqlStatement)
                .parameters(parameters)
                .build();

            // Set data for Movie 2.
            List<AttributeValue> parametersMovie2 = new ArrayList<>();
            AttributeValue attMovie2 = AttributeValue.builder()
                .n("1939")
                .build();

            AttributeValue attMovie2A = AttributeValue.builder()
                .s("Wizard of Oz")
                .build();

            AttributeValue attMovie2B = AttributeValue.builder()
                .s("No Information")
                .build();

            parametersMovie2.add(attMovie2);
            parametersMovie2.add(attMovie2A);
            parametersMovie2.add(attMovie2B);

            BatchStatementRequest statementRequestMovie2 =
BatchStatementRequest.builder()
                .statement(sqlStatement)
                .parameters(parametersMovie2)
                .build();

            // Set data for Movie 3.
            List<AttributeValue> parametersMovie3 = new ArrayList<>();
```

```
            AttributeValue attMovie3 = AttributeValue.builder()
                .n(String.valueOf("2022"))
                .build();

            AttributeValue attMovie3A = AttributeValue.builder()
                .s("My Movie 3")
                .build();

            AttributeValue attMovie3B = AttributeValue.builder()
                .s("No Information")
                .build();

            parametersMovie3.add(attMovie3);
            parametersMovie3.add(attMovie3A);
            parametersMovie3.add(attMovie3B);

            BatchStatementRequest statementRequestMovie3 =
BatchStatementRequest.builder()
                .statement(sqlStatement)
                .parameters(parametersMovie3)
                .build();

            // Add all three movies to the list.
            List<BatchStatementRequest> myBatchStatementList = new ArrayList<>();
            myBatchStatementList.add(statementRequestMovie1);
            myBatchStatementList.add(statementRequestMovie2);
            myBatchStatementList.add(statementRequestMovie3);

            BatchExecuteStatementRequest batchRequest =
BatchExecuteStatementRequest.builder()
                .statements(myBatchStatementList)
                .build();

            BatchExecuteStatementResponse response =
ddb.batchExecuteStatement(batchRequest);
            System.out.println("ExecuteStatement successful: " +
response.toString());
            System.out.println("Added new movies using a batch command.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

```
    public static void updateTableItemBatch(DynamoDbClient ddb) {
        String sqlStatement = "UPDATE MoviesPartiQBatch SET info = 'directors\":
[\"Merian C. Cooper\",\"Ernest B. Schoedsack' where year=? and title=?";
        List<AttributeValue> parametersRec1 = new ArrayList<>();

        // Update three records.
        AttributeValue att1 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("My Movie 1")
            .build();

        parametersRec1.add(att1);
        parametersRec1.add(att2);

        BatchStatementRequest statementRequestRec1 =
 BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec1)
            .build();

        // Update record 2.
        List<AttributeValue> parametersRec2 = new ArrayList<>();
        AttributeValue attRec2 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue attRec2a = AttributeValue.builder()
            .s("My Movie 2")
            .build();

        parametersRec2.add(attRec2);
        parametersRec2.add(attRec2a);
        BatchStatementRequest statementRequestRec2 =
 BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec2)
            .build();

        // Update record 3.
        List<AttributeValue> parametersRec3 = new ArrayList<>();
```

```
        AttributeValue attRec3 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue attRec3a = AttributeValue.builder()
            .s("My Movie 3")
            .build();

        parametersRec3.add(attRec3);
        parametersRec3.add(attRec3a);
        BatchStatementRequest statementRequestRec3 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec3)
            .build();

        // Add all three movies to the list.
        List<BatchStatementRequest> myBatchStatementList = new ArrayList<>();
        myBatchStatementList.add(statementRequestRec1);
        myBatchStatementList.add(statementRequestRec2);
        myBatchStatementList.add(statementRequestRec3);

        BatchExecuteStatementRequest batchRequest =
BatchExecuteStatementRequest.builder()
            .statements(myBatchStatementList)
            .build();

        try {
            BatchExecuteStatementResponse response =
ddb.batchExecuteStatement(batchRequest);
            System.out.println("ExecuteStatement successful: " +
response.toString());
            System.out.println("Updated three movies using a batch command.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Item was updated!");
    }

    public static void deleteItemBatch(DynamoDbClient ddb) {
        String sqlStatement = "DELETE FROM MoviesPartiQBatch WHERE year = ? and
title=?";
```

```
        List<AttributeValue> parametersRec1 = new ArrayList<>();

        // Specify three records to delete.
        AttributeValue att1 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("My Movie 1")
            .build();

        parametersRec1.add(att1);
        parametersRec1.add(att2);

        BatchStatementRequest statementRequestRec1 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec1)
            .build();

        // Specify record 2.
        List<AttributeValue> parametersRec2 = new ArrayList<>();
        AttributeValue attRec2 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();

        AttributeValue attRec2a = AttributeValue.builder()
            .s("My Movie 2")
            .build();

        parametersRec2.add(attRec2);
        parametersRec2.add(attRec2a);
        BatchStatementRequest statementRequestRec2 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec2)
            .build();

        // Specify record 3.
        List<AttributeValue> parametersRec3 = new ArrayList<>();
        AttributeValue attRec3 = AttributeValue.builder()
            .n(String.valueOf("2022"))
            .build();
```

```
        AttributeValue attRec3a = AttributeValue.builder()
            .s("My Movie 3")
            .build();

        parametersRec3.add(attRec3);
        parametersRec3.add(attRec3a);

        BatchStatementRequest statementRequestRec3 =
BatchStatementRequest.builder()
            .statement(sqlStatement)
            .parameters(parametersRec3)
            .build();

        // Add all three movies to the list.
        List<BatchStatementRequest> myBatchStatementList = new ArrayList<>();
        myBatchStatementList.add(statementRequestRec1);
        myBatchStatementList.add(statementRequestRec2);
        myBatchStatementList.add(statementRequestRec3);

        BatchExecuteStatementRequest batchRequest =
BatchExecuteStatementRequest.builder()
            .statements(myBatchStatementList)
            .build();

        try {
            ddb.batchExecuteStatement(batchRequest);
            System.out.println("Deleted three movies using a batch command.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
{
        DeleteTableRequest request = DeleteTableRequest.builder()
            .tableName(tableName)
            .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
```

```
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }

    private static ExecuteStatementResponse
 executeStatementRequest(DynamoDbClient ddb, String statement,

 List<AttributeValue> parameters) {
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
            .statement(statement)
            .parameters(parameters)
            .build();

        return ddb.executeStatement(request);
    }
}
```

- Para obtener información sobre la API, consulte BatchExecuteStatement en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Ejecutar instrucciones PartiQL por lotes.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
```

```
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "Cities";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { type: "confirm", confirmAll },
    );
    const deleteTable = await input.handle({}, { confirmAll });
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
```

```
    }
  }

  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "name",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "name", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");
```

```
  /**
   * Insert items.
   */

  log("Inserting cities into the table.");
  const addItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statements: [
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["Alachua", 10712],
      },
      {
        Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
        Parameters: ["High Springs", 6415],
      },
    ],
  });
  await docClient.send(addItemsStatementCommand);
  log("Cities inserted.");

  /**
   * Select items.
   */

  log("Selecting cities from the table.");
  const selectItemsStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statements: [
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `SELECT * FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
    ],
  });
  const selectItemResponse = await docClient.send(selectItemsStatementCommand);
  log(
```

```
    `Got cities: ${selectItemResponse.Responses.map(
      (r) => `${r.Item.name} (${r.Item.population})`,
    ).join(", ")}`,
  );

  /**
   * Update items.
   */

  log("Modifying the populations.");
  const updateItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statements: [
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [10, "Alachua"],
      },
      {
        Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
        Parameters: [5, "High Springs"],
      },
    ],
  });
  await docClient.send(updateItemStatementCommand);
  log("Updated cities.");

  /**
   * Delete the items.
   */

  log("Deleting the cities.");
  const deleteItemStatementCommand = new BatchExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statements: [
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["Alachua"],
      },
      {
        Statement: `DELETE FROM ${tableName} WHERE name=?`,
        Parameters: ["High Springs"],
      },
```

```
    ],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Cities deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Para obtener información sobre la API, consulte BatchExecuteStatement en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun main() {
    val ddb = DynamoDbClient.fromEnvironment { region = "us-east-1" }
    val tableName = "MoviesPartiQBatch"
    println("Creating an Amazon DynamoDB table named $tableName with a key named
 id and a sort key named title.")
    createTablePartiQLBatch(ddb, tableName, "year")
    putRecordBatch(ddb)
    updateTableItemBatchBatch(ddb)
    deleteItemsBatch(ddb)
    deleteTablePartiQLBatch(tableName)
}
```

```
suspend fun createTablePartiQLBatch(
    ddb: DynamoDbClient,
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }

    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    val response = ddb.createTable(request)
    ddb.waitUntilTableExists {
        // suspend call
        tableName = tableNameVal
    }
    println("The table was successfully created
 ${response.tableDescription?.tableArn}")
```

```
}

suspend fun putRecordBatch(ddb: DynamoDbClient) {
    val sqlStatement = "INSERT INTO MoviesPartiQBatch VALUE {'year':?,
 'title' : ?, 'info' : ?}"

    // Create three movies to add to the Amazon DynamoDB table.
    val parametersMovie1 = mutableListOf<AttributeValue>()
    parametersMovie1.add(AttributeValue.N("2022"))
    parametersMovie1.add(AttributeValue.S("My Movie 1"))
    parametersMovie1.add(AttributeValue.S("No Information"))

    val statementRequestMovie1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie1
        }

    // Set data for Movie 2.
    val parametersMovie2 = mutableListOf<AttributeValue>()
    parametersMovie2.add(AttributeValue.N("2022"))
    parametersMovie2.add(AttributeValue.S("My Movie 2"))
    parametersMovie2.add(AttributeValue.S("No Information"))

    val statementRequestMovie2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie2
        }

    // Set data for Movie 3.
    val parametersMovie3 = mutableListOf<AttributeValue>()
    parametersMovie3.add(AttributeValue.N("2022"))
    parametersMovie3.add(AttributeValue.S("My Movie 3"))
    parametersMovie3.add(AttributeValue.S("No Information"))

    val statementRequestMovie3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersMovie3
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
```

```
    myBatchStatementList.add(statementRequestMovie1)
    myBatchStatementList.add(statementRequestMovie2)
    myBatchStatementList.add(statementRequestMovie3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }
    val response = ddb.batchExecuteStatement(batchRequest)
    println("ExecuteStatement successful: " + response.toString())
    println("Added new movies using a batch command.")
}

suspend fun updateTableItemBatchBatch(ddb: DynamoDbClient) {
    val sqlStatement =
        "UPDATE MoviesPartiQBatch SET info = 'directors\":[\"Merian C. Cooper\",
\"Ernest B. Schoedsack' where year=? and title=?"
    val parametersRec1 = mutableListOf<AttributeValue>()
    parametersRec1.add(AttributeValue.N("2022"))
    parametersRec1.add(AttributeValue.S("My Movie 1"))
    val statementRequestRec1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec1
        }

    // Update record 2.
    val parametersRec2 = mutableListOf<AttributeValue>()
    parametersRec2.add(AttributeValue.N("2022"))
    parametersRec2.add(AttributeValue.S("My Movie 2"))
    val statementRequestRec2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec2
        }

    // Update record 3.
    val parametersRec3 = mutableListOf<AttributeValue>()
    parametersRec3.add(AttributeValue.N("2022"))
    parametersRec3.add(AttributeValue.S("My Movie 3"))
    val statementRequestRec3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec3
```

```
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestRec1)
    myBatchStatementList.add(statementRequestRec2)
    myBatchStatementList.add(statementRequestRec3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }

    val response = ddb.batchExecuteStatement(batchRequest)
    println("ExecuteStatement successful: $response")
    println("Updated three movies using a batch command.")
    println("Items were updated!")
}

suspend fun deleteItemsBatch(ddb: DynamoDbClient) {
    // Specify three records to delete.
    val sqlStatement = "DELETE FROM MoviesPartiQBatch WHERE year = ? and title=?"
    val parametersRec1 = mutableListOf<AttributeValue>()
    parametersRec1.add(AttributeValue.N("2022"))
    parametersRec1.add(AttributeValue.S("My Movie 1"))

    val statementRequestRec1 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec1
        }

    // Specify record 2.
    val parametersRec2 = mutableListOf<AttributeValue>()
    parametersRec2.add(AttributeValue.N("2022"))
    parametersRec2.add(AttributeValue.S("My Movie 2"))
    val statementRequestRec2 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec2
        }

    // Specify record 3.
    val parametersRec3 = mutableListOf<AttributeValue>()
```

```
    parametersRec3.add(AttributeValue.N("2022"))
    parametersRec3.add(AttributeValue.S("My Movie 3"))
    val statementRequestRec3 =
        BatchStatementRequest {
            statement = sqlStatement
            parameters = parametersRec3
        }

    // Add all three movies to the list.
    val myBatchStatementList = mutableListOf<BatchStatementRequest>()
    myBatchStatementList.add(statementRequestRec1)
    myBatchStatementList.add(statementRequestRec2)
    myBatchStatementList.add(statementRequestRec3)

    val batchRequest =
        BatchExecuteStatementRequest {
            statements = myBatchStatementList
        }

    ddb.batchExecuteStatement(batchRequest)
    println("Deleted three movies using a batch command.")
}

suspend fun deleteTablePartiQLBatch(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}
```

- Para obtener información sobre la API, consulte BatchExecuteStatement en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
namespace DynamoDb\PartiQL_Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;

use function AwsUtilities\loadMovieData;
use function AwsUtilities\testable_readline;

class GettingStartedWithPartiQLBatch
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB - PartiQL getting started demo
 using PHP!\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDb\DynamoDBService();

        $tableName = "partiql_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
```

```
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
            $movieYear = testable_readline("Year released: ");
        }
        $key = [
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("INSERT", $tableName, $key);
        $service->insertItemByPartiQLBatch($statement, $parameters);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot),
        ];

        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
```

```
        $service->updateItemByPartiQLBatch($statement, $parameters);
        echo "Movie added and updated.\n";

        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);

        $movie = $service->getItemByPartiQLBatch($tableName, [$key]);
        echo "\nThe movie {$movie['Responses'][0]['Item']['title']['S']}
        was released in {$movie['Responses'][0]['Item']['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Responses'][0]['Item']
['title']['S']}?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot)
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQLBatch($statement, $parameters);

        $movie = $service->getItemByPartiQLBatch($tableName, [$key]);
        echo "Okay, you have rated {$movie['Responses'][0]['Item']['title']
['S']}
        as a {$movie['Responses'][0]['Item']['rating']['N']}\n";

        $service->deleteItemByPartiQLBatch($statement, $parameters);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something
 lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
```

```
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
 born:\n";
        $oops = "Oops! There were no movies released in that year (that we know
 of).\n";
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}

    public function insertItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
```

```
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function getItemByPartiQLBatch(string $tableName, array $keys): Result
    {
        $statements = [];
        foreach ($keys as $key) {
            list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);
            $statements[] = [
                'Statement' => "$statement",
                'Parameters' => $parameters,
            ];
        }

        return $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => $statements,
        ]);
    }

    public function updateItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }

    public function deleteItemByPartiQLBatch(string $statement, array
 $parameters)
    {
        $this->dynamoDbClient->batchExecuteStatement([
            'Statements' => [
                [
```

```
                    'Statement' => "$statement",
                    'Parameters' => $parameters,
                ],
            ],
        ]);
    }
```

- Para obtener información sobre la API, consulte BatchExecuteStatement en la referencia de la API de AWS SDK para PHP.

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear una clase que pueda ejecutar lotes de instrucciones PartiQL.

```
from datetime import datetime
from decimal import Decimal
import logging
from pprint import pprint

import boto3
from botocore.exceptions import ClientError

from scaffold import Scaffold

logger = logging.getLogger(__name__)

class PartiQLBatchWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
```

```
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statements, param_list):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
the
        resource transforms input and output from plain old Python objects
(POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statements: The batch of PartiQL statements.
        :param param_list: The batch of PartiQL parameters that are associated
with
                           each statement. This list must be in the same order as
the
                           statements.
        :return: The responses returned from running the statements, if any.
        """
        try:
            output = self.dyn_resource.meta.client.batch_execute_statement(
                Statements=[
                    {"Statement": statement, "Parameters": params}
                    for statement, params in zip(statements, param_list)
                ]
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute batch of PartiQL statements because the
table "
                    "does not exist."
                )
            else:
                logger.error(
                    "Couldn't execute batch of PartiQL statements. Here's why:
%s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
```

```
            raise
        else:
            return output
```

Ejecutar un escenario que crea una tabla y ejecuta consultas PartiQL en lotes.

```
def run_scenario(scaffold, wrapper, table_name):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB PartiQL batch statement demo.")
    print("-" * 88)

    print(f"Creating table '{table_name}' for the demo...")
    scaffold.create_table(table_name)
    print("-" * 88)

    movie_data = [
        {
            "title": f"House PartiQL",
            "year": datetime.now().year - 5,
            "info": {
                "plot": "Wacky high jinks result from querying a mysterious
 database.",
                "rating": Decimal("8.5"),
            },
        },
        {
            "title": f"House PartiQL 2",
            "year": datetime.now().year - 3,
            "info": {
                "plot": "Moderate high jinks result from querying another
 mysterious database.",
                "rating": Decimal("6.5"),
            },
        },
        {
            "title": f"House PartiQL 3",
            "year": datetime.now().year - 1,
```

```
            "info": {
                "plot": "Tepid high jinks result from querying yet another
mysterious database.",
                "rating": Decimal("2.5"),
            },
        },
    ]

    print(f"Inserting a batch of movies into table '{table_name}.")
    statements = [
        f'INSERT INTO "{table_name}" ' f"VALUE {{'title': ?, 'year': ?,
'info': ?}}"
    ] * len(movie_data)
    params = [list(movie.values()) for movie in movie_data]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Getting data for a batch of movies.")
    statements = [f'SELECT * FROM "{table_name}" WHERE title=? AND year=?'] *
len(
        movie_data
    )
    params = [[movie["title"], movie["year"]] for movie in movie_data]
    output = wrapper.run_partiql(statements, params)
    for item in output["Responses"]:
        print(f"\n{item['Item']['title']}, {item['Item']['year']}")
        pprint(item["Item"])
    print("-" * 88)

    ratings = [Decimal("7.7"), Decimal("5.5"), Decimal("1.3")]
    print(f"Updating a batch of movies with new ratings.")
    statements = [
        f'UPDATE "{table_name}" SET info.rating=? ' f"WHERE title=? AND year=?"
    ] * len(movie_data)
    params = [
        [rating, movie["title"], movie["year"]]
        for rating, movie in zip(ratings, movie_data)
    ]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Getting projected data from the table to verify our update.")
```

```
    output = wrapper.dyn_resource.meta.client.execute_statement(
        Statement=f'SELECT title, info.rating FROM "{table_name}"'
    )
    pprint(output["Items"])
    print("-" * 88)

    print(f"Deleting a batch of movies from the table.")
    statements = [f'DELETE FROM "{table_name}" WHERE title=? AND year=?'] * len(
        movie_data
    )
    params = [[movie["title"], movie["year"]] for movie in movie_data]
    wrapper.run_partiql(statements, params)
    print("Success!")
    print("-" * 88)

    print(f"Deleting table '{table_name}'...")
    scaffold.delete_table()
    print("-" * 88)

    print("\nThanks for watching!")
    print("-" * 88)


if __name__ == "__main__":
    try:
        dyn_res = boto3.resource("dynamodb")
        scaffold = Scaffold(dyn_res)
        movies = PartiQLBatchWrapper(dyn_res)
        run_scenario(scaffold, movies, "doc-example-table-partiql-movies")
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

- Para obtener información sobre la API, consulte BatchWriteItem en Referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Ejecutar un escenario que crea una tabla y ejecuta lotes de consultas PartiQL.

```
 table_name = "doc-example-table-movies-partiql-#{rand(10**4)}"
 scaffold = Scaffold.new(table_name)
 sdk = DynamoDBPartiQLBatch.new(table_name)

 new_step(1, 'Create a new DynamoDB table if none already exists.')
 unless scaffold.exists?(table_name)
   puts("\nNo such table: #{table_name}. Creating it...")
   scaffold.create_table(table_name)
   print "Done!\n".green
 end

 new_step(2, 'Populate DynamoDB table with movie data.')
 download_file = 'moviedata.json'
 puts("Downloading movie database to #{download_file}...")
 movie_data = scaffold.fetch_movie_data(download_file)
 puts("Writing movie data from #{download_file} into your table...")
 scaffold.write_batch(movie_data)
 puts("Records added: #{movie_data.length}.")
 print "Done!\n".green

 new_step(3, 'Select a batch of items from the movies table.')
 puts "Let's select some popular movies for side-by-side comparison."
 response = sdk.batch_execute_select([['Mean Girls', 2004], ['Goodfellas',
1977], ['The Prancing of the Lambs', 2005]])
 puts("Items selected: #{response['responses'].length}\n")
 print "\nDone!\n".green

 new_step(4, 'Delete a batch of items from the movies table.')
 sdk.batch_execute_write([['Mean Girls', 2004], ['Goodfellas', 1977], ['The
Prancing of the Lambs', 2005]])
 print "\nDone!\n".green
```

```
  new_step(5, 'Delete the table.')
  return unless scaffold.exists?(table_name)

  scaffold.delete_table
end
```

- Para obtener información sobre la API, consulte BatchExecuteStatement en la referencia de la API de AWS SDK para Ruby.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Consulta de una tabla de DynamoDB con PartiQL y un SDK de AWS

En el siguiente ejemplo de código, se muestra cómo:

- Obtener un artículo mediante una instrucción SELECT.
- Agregar un elemento mediante una instrucción INSERT.
- Actualizar un elemento mediante una instrucción UPDATE.
- Eliminar un elemento mediante una instrucción DELETE.

.NET

SDK para .NET

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
namespace PartiQL_Basics_Scenario
{
    public class PartiQLMethods
    {
```

```
        private static readonly AmazonDynamoDBClient Client = new
AmazonDynamoDBClient();


        /// <summary>
        /// Inserts movies imported from a JSON file into the movie table by
        /// using an Amazon DynamoDB PartiQL INSERT statement.
        /// </summary>
        /// <param name="tableName">The name of the table where the movie
        /// information will be inserted.</param>
        /// <param name="movieFileName">The name of the JSON file that contains
        /// movie information.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the insert operation.</returns>
        public static async Task<bool> InsertMovies(string tableName, string
movieFileName)
        {
            // Get the list of movies from the JSON file.
            var movies = ImportMovies(movieFileName);

            var success = false;

            if (movies is not null)
            {
                // Insert the movies in a batch using PartiQL. Because the
                // batch can contain a maximum of 25 items, insert 25 movies
                // at a time.
                string insertBatch = $"INSERT INTO {tableName} VALUE
{{'title': ?, 'year': ?}}";
                var statements = new List<BatchStatementRequest>();

                try
                {
                    for (var indexOffset = 0; indexOffset < 250; indexOffset +=
25)
                    {
                        for (var i = indexOffset; i < indexOffset + 25; i++)
                        {
                            statements.Add(new BatchStatementRequest
                            {
                                Statement = insertBatch,
                                Parameters = new List<AttributeValue>
                                {
                                    new AttributeValue { S = movies[i].Title },
```

```
                                    new AttributeValue { N =
movies[i].Year.ToString() },
                                },
                            });
                        }

                        var response = await
Client.BatchExecuteStatementAsync(new BatchExecuteStatementRequest
                        {
                            Statements = statements,
                        });

                        // Wait between batches for movies to be successfully
added.
                        System.Threading.Thread.Sleep(3000);

                        success = response.HttpStatusCode ==
System.Net.HttpStatusCode.OK;

                        // Clear the list of statements for the next batch.
                        statements.Clear();
                    }
                }
                catch (AmazonDynamoDBException ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }

            return success;
        }

        /// <summary>
        /// Loads the contents of a JSON file into a list of movies to be
        /// added to the DynamoDB table.
        /// </summary>
        /// <param name="movieFileName">The full path to the JSON file.</param>
        /// <returns>A generic list of movie objects.</returns>
        public static List<Movie> ImportMovies(string movieFileName)
        {
            if (!File.Exists(movieFileName))
            {
                return null!;
            }
```

```
            using var sr = new StreamReader(movieFileName);
            string json = sr.ReadToEnd();
            var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

            if (allMovies is not null)
            {
                // Return the first 250 entries.
                return allMovies.GetRange(0, 250);
            }
            else
            {
                return null!;
            }
        }



        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

```
        /// <summary>
        /// Retrieve multiple movies by year using a SELECT statement.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="year">The year the movies were released.</param>
        /// <returns></returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
GetMovies(string tableName, int year)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE year = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { N = year.ToString() },
            };

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }


        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";
```

```
            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });
```

```
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Displays the list of movies returned from a database query.
        /// </summary>
        /// <param name="items">The list of movie information to display.</param>
        private static void DisplayMovies(List<Dictionary<string,
AttributeValue>> items)
        {
            if (items.Count > 0)
            {
                Console.WriteLine($"Found {items.Count} movies.");
```

```
                items.ForEach(item =>
 Console.WriteLine($"{item["year"].N}\t{item["title"].S}"));
            }
            else
            {
                Console.WriteLine($"Didn't find a movie that matched the supplied
 criteria.");
            }
        }


    }
}



        /// <summary>
        /// Uses a PartiQL SELECT statement to retrieve a single movie from the
        /// movie database.
        /// </summary>
        /// <param name="tableName">The name of the movie table.</param>
        /// <param name="movieTitle">The title of the movie to retrieve.</param>
        /// <returns>A list of movie data. If no movie matches the supplied
        /// title, the list is empty.</returns>
        public static async Task<List<Dictionary<string, AttributeValue>>>
 GetSingleMovie(string tableName, string movieTitle)
        {
            string selectSingle = $"SELECT * FROM {tableName} WHERE title = ?";
            var parameters = new List<AttributeValue>
            {
                new AttributeValue { S = movieTitle },
            };

            var response = await Client.ExecuteStatementAsync(new
 ExecuteStatementRequest
            {
                Statement = selectSingle,
                Parameters = parameters,
            });

            return response.Items;
        }
```

```
        /// <summary>
        /// Inserts a single movie into the movies table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to insert.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success or failure of
        /// the INSERT operation.</returns>
        public static async Task<bool> InsertSingleMovie(string tableName, string
movieTitle, int year)
        {
            string insertBatch = $"INSERT INTO {tableName} VALUE {{'title': ?,
'year': ?}}";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertBatch,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Updates a single movie in the table, adding information for the
        /// producer.
        /// </summary>
        /// <param name="tableName">the name of the table.</param>
        /// <param name="producer">The name of the producer.</param>
        /// <param name="movieTitle">The movie title.</param>
        /// <param name="year">The year the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// UPDATE operation.</returns>
        public static async Task<bool> UpdateSingleMovie(string tableName, string
producer, string movieTitle, int year)
        {
```

```
            string insertSingle = $"UPDATE {tableName} SET Producer=? WHERE title
= ? AND year = ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = insertSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = producer },
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }



        /// <summary>
        /// Deletes a single movie from the table.
        /// </summary>
        /// <param name="tableName">The name of the table.</param>
        /// <param name="movieTitle">The title of the movie to delete.</param>
        /// <param name="year">The year that the movie was released.</param>
        /// <returns>A Boolean value that indicates the success of the
        /// DELETE operation.</returns>
        public static async Task<bool> DeleteSingleMovie(string tableName, string
movieTitle, int year)
        {
            var deleteSingle = $"DELETE FROM {tableName} WHERE title = ? AND year
= ?";

            var response = await Client.ExecuteStatementAsync(new
ExecuteStatementRequest
            {
                Statement = deleteSingle,
                Parameters = new List<AttributeValue>
                {
                    new AttributeValue { S = movieTitle },
                    new AttributeValue { N = year.ToString() },
                },
            });
```

```
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

- Para obtener información sobre la API, consulte ExecuteStatement en la referencia de la API de AWS SDK para .NET.

C++

SDK para C++

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
        //  1. Create a table. (CreateTable)
        if (AwsDoc::DynamoDB::createMoviesDynamoDBTable(clientConfig)) {

            AwsDoc::DynamoDB::partiqlExecuteScenario(clientConfig);

            // 7. Delete the table. (DeleteTable)
            AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(clientConfig);
        }

//! Scenario to modify and query a DynamoDB table using single PartiQL
 statements.
/*!
  \sa partiqlExecuteScenario()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::DynamoDB::partiqlExecuteScenario(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    // 2. Add a new movie using an "Insert" statement. (ExecuteStatement)
```

```
   Aws::String title;
   float rating;
   int year;
   Aws::String plot;
   {
       title = askQuestion(
               "Enter the title of a movie you want to add to the table: ");
       year = askQuestionForInt("What year was it released? ");
       rating = askQuestionForFloatRange("On a scale of 1 - 10, how do you rate
it? ",
                                         1, 10);
       plot = askQuestion("Summarize the plot for me: ");

       Aws::DynamoDB::Model::ExecuteStatementRequest request;
       std::stringstream sqlStream;
       sqlStream << "INSERT INTO \"" << MOVIE_TABLE_NAME << "\" VALUE {'"
                 << TITLE_KEY << "': ?, '" << YEAR_KEY << "': ?, '"
                 << INFO_KEY << "': ?}";

       request.SetStatement(sqlStream.str());

       // Create the parameter attributes.
       Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
       attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

       Aws::DynamoDB::Model::AttributeValue infoMapAttribute;

       std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> ratingAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
               ALLOCATION_TAG.c_str());
       ratingAttribute->SetN(rating);
       infoMapAttribute.AddMEntry(RATING_KEY, ratingAttribute);

       std::shared_ptr<Aws::DynamoDB::Model::AttributeValue> plotAttribute =
Aws::MakeShared<Aws::DynamoDB::Model::AttributeValue>(
               ALLOCATION_TAG.c_str());
       plotAttribute->SetS(plot);
       infoMapAttribute.AddMEntry(PLOT_KEY, plotAttribute);
       attributes.push_back(infoMapAttribute);
       request.SetParameters(attributes);

       Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
```

```
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to add a movie: " <<
outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    std::cout << "\nAdded '" << title << "' to '" << MOVIE_TABLE_NAME << "'."
              << std::endl;

    //  3. Get the data for the movie using a "Select" statement.
(ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to retrieve movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
        else {
            // Print the retrieved movie information.
            const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();
```

```
            if (items.size() == 1) {
                printMovieInfo(items[0]);
            }
            else {
                std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                          << " There should be only one movie." << std::endl;
            }
        }
    }

    //  4. Update the data for the movie using an "Update" statement.
(ExecuteStatement)
    {
        rating = askQuestionForFloatRange(
                Aws::String("\nLet's update your movie.\nYou rated it  ") +
                std::to_string(rating)
                + ", what new rating would you give it? ", 1, 10);

        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "UPDATE \"" << MOVIE_TABLE_NAME << "\" SET "
                  << INFO_KEY << "." << RATING_KEY << "=? WHERE "
                  << TITLE_KEY << "=? AND " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;

attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(rating));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));

        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);

        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to update a movie: "
                      << outcome.GetError().GetMessage();
            return false;
```

```
        }
    }

    std::cout << "\nUpdated '" << title << "' with new attributes:" << std::endl;

    //  5. Get the updated data for the movie using a "Select" statement.
(ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "SELECT * FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
dynamoClient.ExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to retrieve the movie information: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
        else {
            const Aws::DynamoDB::Model::ExecuteStatementResult &result =
outcome.GetResult();

            const Aws::Vector<Aws::Map<Aws::String,
Aws::DynamoDB::Model::AttributeValue>> &items = result.GetItems();

            if (items.size() == 1) {
                printMovieInfo(items[0]);
            }
            else {
                std::cerr << "Error: " << items.size() << " movies were
retrieved. "
                          << " There should be only one movie." << std::endl;
            }
        }
```

```
    }

    std::cout << "Deleting the movie" << std::endl;

    // 6. Delete the movie using a "Delete" statement. (ExecuteStatement)
    {
        Aws::DynamoDB::Model::ExecuteStatementRequest request;
        std::stringstream sqlStream;
        sqlStream << "DELETE FROM  \"" << MOVIE_TABLE_NAME << "\" WHERE "
                  << TITLE_KEY << "=? and " << YEAR_KEY << "=?";

        request.SetStatement(sqlStream.str());

        Aws::Vector<Aws::DynamoDB::Model::AttributeValue> attributes;
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetS(title));
        attributes.push_back(Aws::DynamoDB::Model::AttributeValue().SetN(year));
        request.SetParameters(attributes);

        Aws::DynamoDB::Model::ExecuteStatementOutcome outcome =
 dynamoClient.ExecuteStatement(
                request);
        if (!outcome.IsSuccess()) {
            std::cerr << "Failed to delete the movie: "
                      << outcome.GetError().GetMessage() << std::endl;
            return false;
        }
    }

    std::cout << "Movie successfully deleted." << std::endl;
    return true;
}

//! Create a DynamoDB table to be used in sample code scenarios.
/*!
  \sa createMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::createMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    bool movieTableAlreadyExisted = false;
```

```
    {
        Aws::DynamoDB::Model::CreateTableRequest request;

        Aws::DynamoDB::Model::AttributeDefinition yearAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(YEAR_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::N);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::AttributeDefinition titleAttributeDefinition;
        yearAttributeDefinition.SetAttributeName(TITLE_KEY);
        yearAttributeDefinition.SetAttributeType(
                Aws::DynamoDB::Model::ScalarAttributeType::S);
        request.AddAttributeDefinitions(yearAttributeDefinition);

        Aws::DynamoDB::Model::KeySchemaElement yearKeySchema;
        yearKeySchema.WithAttributeName(YEAR_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::HASH);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::KeySchemaElement titleKeySchema;
        yearKeySchema.WithAttributeName(TITLE_KEY).WithKeyType(
                Aws::DynamoDB::Model::KeyType::RANGE);
        request.AddKeySchema(yearKeySchema);

        Aws::DynamoDB::Model::ProvisionedThroughput throughput;
        throughput.WithReadCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS).WithWriteCapacityUnits(
                PROVISIONED_THROUGHPUT_UNITS);
        request.SetProvisionedThroughput(throughput);
        request.SetTableName(MOVIE_TABLE_NAME);

        std::cout << "Creating table '" << MOVIE_TABLE_NAME << "'..." <<
std::endl;
        const Aws::DynamoDB::Model::CreateTableOutcome &result =
dynamoClient.CreateTable(
                request);
        if (!result.IsSuccess()) {
            if (result.GetError().GetErrorType() ==
                Aws::DynamoDB::DynamoDBErrors::RESOURCE_IN_USE) {
                std::cout << "Table already exists." << std::endl;
                movieTableAlreadyExisted = true;
            }
            else {
```

```
                std::cerr << "Failed to create table: "
                          << result.GetError().GetMessage();
                return false;
            }
        }
    }

    // Wait for table to become active.
    if (!movieTableAlreadyExisted) {
        std::cout << "Waiting for table '" << MOVIE_TABLE_NAME
                  << "' to become active...." << std::endl;
        if (!AwsDoc::DynamoDB::waitTableActive(MOVIE_TABLE_NAME,
 clientConfiguration)) {
            return false;
        }
        std::cout << "Table '" << MOVIE_TABLE_NAME << "' created and active."
                  << std::endl;
    }

    return true;
}

//! Delete the DynamoDB table used for sample code scenarios.
/*!
  \sa deleteMoviesDynamoDBTable()
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::deleteMoviesDynamoDBTable(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::DynamoDB::DynamoDBClient dynamoClient(clientConfiguration);

    Aws::DynamoDB::Model::DeleteTableRequest request;
    request.SetTableName(MOVIE_TABLE_NAME);

    const Aws::DynamoDB::Model::DeleteTableOutcome &result =
 dynamoClient.DeleteTable(
            request);
    if (result.IsSuccess()) {
        std::cout << "Your table \""
                  << result.GetResult().GetTableDescription().GetTableName()
                  << " was deleted.\n";
    }
    else {
```

```
        std::cerr << "Failed to delete table: " << result.GetError().GetMessage()
                  << std::endl;
    }

    return result.IsSuccess();
}

//! Query a newly created DynamoDB table until it is active.
/*!
  \sa waitTableActive()
  \param waitTableActive: The DynamoDB table's name.
  \param dynamoClient: A DynamoDB client.
  \return bool: Function succeeded.
*/
bool AwsDoc::DynamoDB::waitTableActive(const Aws::String &tableName,
                                       const Aws::DynamoDB::DynamoDBClient
 &dynamoClient) {

    // Repeatedly call DescribeTable until table is ACTIVE.
    const int MAX_QUERIES = 20;
    Aws::DynamoDB::Model::DescribeTableRequest request;
    request.SetTableName(tableName);

    int count = 0;
    while (count < MAX_QUERIES) {
        const Aws::DynamoDB::Model::DescribeTableOutcome &result =
 dynamoClient.DescribeTable(
                request);
        if (result.IsSuccess()) {
            Aws::DynamoDB::Model::TableStatus status =
 result.GetResult().GetTable().GetTableStatus();

            if (Aws::DynamoDB::Model::TableStatus::ACTIVE != status) {
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
            else {
                return true;
            }
        }
        else {
            std::cerr << "Error DynamoDB::waitTableActive "
                      << result.GetError().GetMessage() << std::endl;
            return false;
        }
```

```
        count++;
    }
    return false;
}
```

- Para obtener información sobre la API, consulte ExecuteStatement en la referencia de la API de AWS SDK para C++.

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Ejecutar un escenario que crea una tabla y ejecuta consultas PartiQL.

```
import (
 "context"
 "fmt"
 "log"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunPartiQLSingleScenario shows you how to use the AWS SDK for Go
// to use PartiQL to query a table that stores data about movies.
//
// * Use PartiQL statements to add, get, update, and delete data for individual
 movies.
//
// This example creates an Amazon DynamoDB service client from the specified
 sdkConfig so that
```

```
// you can replace it with a mocked or stubbed config for unit testing.
//
// This example creates and deletes a DynamoDB table to use during the scenario.
func RunPartiQLSingleScenario(ctx context.Context, sdkConfig aws.Config,
 tableName string) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB PartiQL single action demo.")
 log.Println(strings.Repeat("-", 88))

 tableBasics := actions.TableBasics{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }
 runner := actions.PartiQLRunner{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }

 exists, err := tableBasics.TableExists(ctx)
 if err != nil {
  panic(err)
 }
 if !exists {
  log.Printf("Creating table %v...\n", tableName)
  _, err = tableBasics.CreateMovieTable(ctx)
  if err != nil {
   panic(err)
  } else {
   log.Printf("Created table %v.\n", tableName)
  }
 } else {
  log.Printf("Table %v already exists.\n", tableName)
 }
 log.Println(strings.Repeat("-", 88))

 currentYear, _, _ := time.Now().Date()
 customMovie := actions.Movie{
  Title: "24 Hour PartiQL People",
```

```
 Year:  currentYear,
 Info: map[string]interface{}{
  "plot":   "A group of data developers discover a new query language they can't
stop using.",
  "rating": 9.9,
 },
}

log.Printf("Inserting movie '%v' released in %v.", customMovie.Title,
customMovie.Year)
err = runner.AddMovie(ctx, customMovie)
if err == nil {
 log.Printf("Added %v to the movie table.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Getting data for movie '%v' released in %v.", customMovie.Title,
customMovie.Year)
movie, err := runner.GetMovie(ctx, customMovie.Title, customMovie.Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))

newRating := 6.6
log.Printf("Updating movie '%v' with a rating of %v.", customMovie.Title,
newRating)
err = runner.UpdateMovie(ctx, customMovie, newRating)
if err == nil {
 log.Printf("Updated %v with a new rating.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Getting data again to verify the update.")
movie, err = runner.GetMovie(ctx, customMovie.Title, customMovie.Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Deleting movie '%v'.\n", customMovie.Title)
err = runner.DeleteMovie(ctx, customMovie)
if err == nil {
 log.Printf("Deleted %v.\n", customMovie.Title)
```

```
 }

 err = tableBasics.DeleteTable(ctx)
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Defina una estructura Película para utilizar en este ejemplo.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite
 primary key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
```

```
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Crear una estructura y métodos que ejecuten instrucciones PartiQL.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

```
// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't insert an item with PartiQL. Here's why: %v\n", err)
 }
 return err
}



// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB
 table by
// title and year.
func (runner PartiQLRunner) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 var movie Movie
 params, err := attributevalue.MarshalList([]interface{}{title, year})
 if err != nil {
  panic(err)
 }
 response, err := runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
```

```
 } else {
  err = attributevalue.UnmarshalMap(response.Items[0], &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}



// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie
 that
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(ctx context.Context, movie Movie, rating
 float64) error {
 params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
 }
 return err
}



// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the
 DynamoDB table.
func (runner PartiQLRunner) DeleteMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
```

```
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

- Para obtener información sobre la API, consulte ExecuteStatement en la referencia de la API de AWS SDK para Go.

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
public class ScenarioPartiQ {
    public static void main(String[] args) throws IOException {
        String fileName = "../../../resources/sample_files/movies.json";
        String tableName = "MoviesPartiQ";
        Region region = Region.US_EAST_1;
        DynamoDbClient ddb = DynamoDbClient.builder()
            .region(region)
            .build();

        System.out.println(
            "******* Creating an Amazon DynamoDB table named MoviesPartiQ with a
 key named year and a sort key named title.");
```

```
        createTable(ddb, tableName);

        System.out.println("Loading data into the MoviesPartiQ table.");
        loadData(ddb, fileName);

        System.out.println("Getting data from the MoviesPartiQ table.");
        getItem(ddb);

        System.out.println("Putting a record into the MoviesPartiQ table.");
        putRecord(ddb);

        System.out.println("Updating a record.");
        updateTableItem(ddb);

        System.out.println("Querying the movies released in 2013.");
        queryTable(ddb);

        System.out.println("Deleting the Amazon DynamoDB table.");
        deleteDynamoDBTable(ddb, tableName);
        ddb.close();
    }

    public static void createTable(DynamoDbClient ddb, String tableName) {
        DynamoDbWaiter dbWaiter = ddb.waiter();
        ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<>();

        // Define attributes.
        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("year")
            .attributeType("N")
            .build());

        attributeDefinitions.add(AttributeDefinition.builder()
            .attributeName("title")
            .attributeType("S")
            .build());

        ArrayList<KeySchemaElement> tableKey = new ArrayList<>();
        KeySchemaElement key = KeySchemaElement.builder()
            .attributeName("year")
            .keyType(KeyType.HASH)
            .build();

        KeySchemaElement key2 = KeySchemaElement.builder()
```

```
            .attributeName("title")
            .keyType(KeyType.RANGE) // Sort
            .build();

        // Add KeySchemaElement objects to the list.
        tableKey.add(key);
        tableKey.add(key2);

        CreateTableRequest request = CreateTableRequest.builder()
            .keySchema(tableKey)
            .billingMode(BillingMode.PAY_PER_REQUEST) //Scales based on traffic.
            .attributeDefinitions(attributeDefinitions)
            .tableName(tableName)
            .build();

        try {
            CreateTableResponse response = ddb.createTable(request);
            DescribeTableRequest tableRequest = DescribeTableRequest.builder()
                .tableName(tableName)
                .build();

            // Wait until the Amazon DynamoDB table is created.
            WaiterResponse<DescribeTableResponse> waiterResponse =
dbWaiter.waitUntilTableExists(tableRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            String newTable = response.tableDescription().tableName();
            System.out.println("The " + newTable + " was successfully created.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    // Load data into the table.
    public static void loadData(DynamoDbClient ddb, String fileName) throws
IOException {

        String sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?,
'title' : ?, 'info' : ?}";
        JsonParser parser = new JsonFactory().createParser(new File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        Iterator<JsonNode> iter = rootNode.iterator();
```

```
        ObjectNode currentNode;
        int t = 0;
        List<AttributeValue> parameters = new ArrayList<>();
        while (iter.hasNext()) {

            // Add 200 movies to the table.
            if (t == 200)
                break;
            currentNode = (ObjectNode) iter.next();

            int year = currentNode.path("year").asInt();
            String title = currentNode.path("title").asText();
            String info = currentNode.path("info").toString();

            AttributeValue att1 = AttributeValue.builder()
                .n(String.valueOf(year))
                .build();

            AttributeValue att2 = AttributeValue.builder()
                .s(title)
                .build();

            AttributeValue att3 = AttributeValue.builder()
                .s(info)
                .build();

            parameters.add(att1);
            parameters.add(att2);
            parameters.add(att3);

            // Insert the movie into the Amazon DynamoDB table.
            executeStatementRequest(ddb, sqlStatement, parameters);
            System.out.println("Added Movie " + title);

            parameters.remove(att1);
            parameters.remove(att2);
            parameters.remove(att3);
            t++;
        }
    }

    public static void getItem(DynamoDbClient ddb) {
```

```
        String sqlStatement = "SELECT * FROM MoviesPartiQ where year=? and
title=?";
        List<AttributeValue> parameters = new ArrayList<>();
        AttributeValue att1 = AttributeValue.builder()
            .n("2012")
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("The Perks of Being a Wallflower")
            .build();

        parameters.add(att1);
        parameters.add(att2);

        try {
            ExecuteStatementResponse response = executeStatementRequest(ddb,
sqlStatement, parameters);
            System.out.println("ExecuteStatement successful: " +
response.toString());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void putRecord(DynamoDbClient ddb) {

        String sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?,
'title' : ?, 'info' : ?}";
        try {
            List<AttributeValue> parameters = new ArrayList<>();

            AttributeValue att1 = AttributeValue.builder()
                .n(String.valueOf("2020"))
                .build();

            AttributeValue att2 = AttributeValue.builder()
                .s("My Movie")
                .build();

            AttributeValue att3 = AttributeValue.builder()
                .s("No Information")
                .build();
```

```
            parameters.add(att1);
            parameters.add(att2);
            parameters.add(att3);

            executeStatementRequest(ddb, sqlStatement, parameters);
            System.out.println("Added new movie.");

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void updateTableItem(DynamoDbClient ddb) {

        String sqlStatement = "UPDATE MoviesPartiQ SET info = 'directors\":
[\"Merian C. Cooper\",\"Ernest B. Schoedsack' where year=? and title=?";
        List<AttributeValue> parameters = new ArrayList<>();
        AttributeValue att1 = AttributeValue.builder()
            .n(String.valueOf("2013"))
            .build();

        AttributeValue att2 = AttributeValue.builder()
            .s("The East")
            .build();

        parameters.add(att1);
        parameters.add(att2);

        try {
            executeStatementRequest(ddb, sqlStatement, parameters);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Item was updated!");
    }

    // Query the table where the year is 2013.
    public static void queryTable(DynamoDbClient ddb) {
        String sqlStatement = "SELECT * FROM MoviesPartiQ where year = ? ORDER BY
 year";
```

```
        try {

            List<AttributeValue> parameters = new ArrayList<>();
            AttributeValue att1 = AttributeValue.builder()
                .n(String.valueOf("2013"))
                .build();
            parameters.add(att1);

            // Get items in the table and write out the ID value.
            ExecuteStatementResponse response = executeStatementRequest(ddb,
sqlStatement, parameters);
            System.out.println("ExecuteStatement successful: " +
response.toString());

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void deleteDynamoDBTable(DynamoDbClient ddb, String tableName)
{

        DeleteTableRequest request = DeleteTableRequest.builder()
            .tableName(tableName)
            .build();

        try {
            ddb.deleteTable(request);

        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println(tableName + " was successfully deleted!");
    }

    private static ExecuteStatementResponse
executeStatementRequest(DynamoDbClient ddb, String statement,

List<AttributeValue> parameters) {
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
            .statement(statement)
            .parameters(parameters)
```

```
            .build();

        return ddb.executeStatement(request);
    }

    private static void processResults(ExecuteStatementResponse
 executeStatementResult) {
        System.out.println("ExecuteStatement successful: " +
 executeStatementResult.toString());
    }
}
```

- Para obtener información sobre la API, consulte ExecuteStatement en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Ejecutar instrucciones PartiQL individuales.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";
```

```
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "SingleOriginCoffees";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { type: "confirm", confirmAll },
    );
    const deleteTable = await input.handle({});
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
 table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }

  /**
   * Create a table.
   */
```

```
  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "varietal",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "varietal", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert an item.
   */

  log("Inserting a coffee into the table.");
  const addItemStatementCommand = new ExecuteStatementCommand({
```

```
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
    Statement: `INSERT INTO ${tableName} value {'varietal':?, 'profile':?}`,
    Parameters: ["arabica", ["chocolate", "floral"]],
  });
  await client.send(addItemStatementCommand);
  log("Coffee inserted.");

  /**
   * Select an item.
   */

  log("Selecting the coffee from the table.");
  const selectItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statement: `SELECT * FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  const selectItemResponse = await docClient.send(selectItemStatementCommand);
  log(`Got coffee: ${JSON.stringify(selectItemResponse.Items[0])}`);

  /**
   * Update the item.
   */

  log("Add a flavor profile to the coffee.");
  const updateItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statement: `UPDATE ${tableName} SET profile=list_append(profile, ?) WHERE
 varietal=?`,
    Parameters: [["fruity"], "arabica"],
  });
  await client.send(updateItemStatementCommand);
  log("Updated coffee");

  /**
   * Delete the item.
   */

  log("Deleting the coffee.");
  const deleteItemStatementCommand = new ExecuteStatementCommand({
```

```
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statement: `DELETE FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Coffee deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};
```

- Para obtener información sobre la API, consulte ExecuteStatement en la referencia de la API de AWS SDK para JavaScript.

Kotlin

SDK para Kotlin

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
suspend fun main() {
    val ddb = DynamoDbClient.fromEnvironment { region = "us-east-1" }
    val tableName = "MoviesPartiQ"
    val fileName = "../../../resources/sample_files/movies.json"
    println("Creating an Amazon DynamoDB table named MoviesPartiQ with a key
 named id and a sort key named title.")
    createTablePartiQL(ddb, tableName, "year")
    loadDataPartiQL(ddb, fileName)
```

```
    println("******* Getting data from the MoviesPartiQ table.")
    getMoviePartiQL(ddb)

    println("******* Putting a record into the MoviesPartiQ table.")
    putRecordPartiQL(ddb)

    println("******* Updating a record.")
    updateTableItemPartiQL(ddb)

    println("******* Querying the movies released in 2013.")
    queryTablePartiQL(ddb)

    println("******* Deleting the MoviesPartiQ table.")
    deleteTablePartiQL(tableName)
}

suspend fun createTablePartiQL(
    ddb: DynamoDbClient,
    tableNameVal: String,
    key: String,
) {
    val attDef =
        AttributeDefinition {
            attributeName = key
            attributeType = ScalarAttributeType.N
        }

    val attDef1 =
        AttributeDefinition {
            attributeName = "title"
            attributeType = ScalarAttributeType.S
        }

    val keySchemaVal =
        KeySchemaElement {
            attributeName = key
            keyType = KeyType.Hash
        }

    val keySchemaVal1 =
        KeySchemaElement {
            attributeName = "title"
            keyType = KeyType.Range
        }
```

```
    val request =
        CreateTableRequest {
            attributeDefinitions = listOf(attDef, attDef1)
            keySchema = listOf(keySchemaVal, keySchemaVal1)
            billingMode = BillingMode.PayPerRequest
            tableName = tableNameVal
        }

    val response = ddb.createTable(request)
    ddb.waitUntilTableExists {
        // suspend call
        tableName = tableNameVal
    }
    println("The table was successfully created
 ${response.tableDescription?.tableArn}")
}

suspend fun loadDataPartiQL(
    ddb: DynamoDbClient,
    fileName: String,
) {
    val sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?, 'title' : ?,
 'info' : ?}"
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode
    var t = 0

    while (iter.hasNext()) {
        if (t == 200) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()

        val parameters: MutableList<AttributeValue> = ArrayList<AttributeValue>()
        parameters.add(AttributeValue.N(year.toString()))
        parameters.add(AttributeValue.S(title))
        parameters.add(AttributeValue.S(info))
```

```
        executeStatementPartiQL(ddb, sqlStatement, parameters)
        println("Added Movie $title")
        parameters.clear()
        t++
    }
}

suspend fun getMoviePartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "SELECT * FROM MoviesPartiQ where year=? and title=?"
    val parameters: MutableList<AttributeValue> = ArrayList<AttributeValue>()
    parameters.add(AttributeValue.N("2012"))
    parameters.add(AttributeValue.S("The Perks of Being a Wallflower"))
    val response = executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("ExecuteStatement successful: $response")
}

suspend fun putRecordPartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "INSERT INTO MoviesPartiQ VALUE {'year':?, 'title' : ?,
 'info' : ?}"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2020"))
    parameters.add(AttributeValue.S("My Movie"))
    parameters.add(AttributeValue.S("No Info"))
    executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("Added new movie.")
}

suspend fun updateTableItemPartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "UPDATE MoviesPartiQ SET info = 'directors\":[\"Merian C.
 Cooper\",\"Ernest B. Schoedsack\' where year=? and title=?"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2013"))
    parameters.add(AttributeValue.S("The East"))
    executeStatementPartiQL(ddb, sqlStatement, parameters)
    println("Item was updated!")
}

// Query the table where the year is 2013.
suspend fun queryTablePartiQL(ddb: DynamoDbClient) {
    val sqlStatement = "SELECT * FROM MoviesPartiQ where year = ?"
    val parameters: MutableList<AttributeValue> = java.util.ArrayList()
    parameters.add(AttributeValue.N("2013"))
    val response = executeStatementPartiQL(ddb, sqlStatement, parameters)
```

```
    println("ExecuteStatement successful: $response")
}

suspend fun deleteTablePartiQL(tableNameVal: String) {
    val request =
        DeleteTableRequest {
            tableName = tableNameVal
        }

    DynamoDbClient { region = "us-east-1" }.use { ddb ->
        ddb.deleteTable(request)
        println("$tableNameVal was deleted")
    }
}

suspend fun executeStatementPartiQL(
    ddb: DynamoDbClient,
    statementVal: String,
    parametersVal: List<AttributeValue>,
): ExecuteStatementResponse {
    val request =
        ExecuteStatementRequest {
            statement = statementVal
            parameters = parametersVal
        }

    return ddb.executeStatement(request)
}
```

- Para obtener información sobre la API, consulte ExecuteStatement en la referencia de la API de AWS SDK para Kotlin.

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
namespace DynamoDb\PartiQL_Basics;

use Aws\DynamoDb\Marshaler;
use DynamoDb;
use DynamoDb\DynamoDBAttribute;

use function AwsUtilities\testable_readline;
use function AwsUtilities\loadMovieData;

class GettingStartedWithPartiQL
{
    public function run()
    {
        echo("\n");
        echo("--------------------------------------\n");
        print("Welcome to the Amazon DynamoDB - PartiQL getting started demo
 using PHP!\n");
        echo("--------------------------------------\n");

        $uuid = uniqid();
        $service = new DynamoDb\DynamoDBService();

        $tableName = "partiql_demo_table_$uuid";
        $service->createTable(
            $tableName,
            [
                new DynamoDBAttribute('year', 'N', 'HASH'),
                new DynamoDBAttribute('title', 'S', 'RANGE')
            ]
        );

        echo "Waiting for table...";
        $service->dynamoDbClient->waitUntil("TableExists", ['TableName' =>
 $tableName]);
        echo "table $tableName found!\n";

        echo "What's the name of the last movie you watched?\n";
        while (empty($movieName)) {
            $movieName = testable_readline("Movie name: ");
        }
        echo "And what year was it released?\n";
        $movieYear = "year";
        while (!is_numeric($movieYear) || intval($movieYear) != $movieYear) {
```

```
            $movieYear = testable_readline("Year released: ");
        }
        $key = [
            'Item' => [
                'year' => [
                    'N' => "$movieYear",
                ],
                'title' => [
                    'S' => $movieName,
                ],
            ],
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("INSERT", $tableName, $key);
        $service->insertItemByPartiQL($statement, $parameters);

        echo "How would you rate the movie from 1-10?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        echo "What was the movie about?\n";
        while (empty($plot)) {
            $plot = testable_readline("Plot summary: ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot),
        ];

        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQL($statement, $parameters);
        echo "Movie added and updated.\n";



        $batch = json_decode(loadMovieData());

        $service->writeBatch($tableName, $batch);

        $movie = $service->getItemByPartiQL($tableName, $key);
```

```
        echo "\nThe movie {$movie['Items'][0]['title']['S']} was released in
 {$movie['Items'][0]['year']['N']}.\n";
        echo "What rating would you like to give {$movie['Items'][0]['title']
['S']}?\n";
        $rating = 0;
        while (!is_numeric($rating) || intval($rating) != $rating || $rating < 1
 || $rating > 10) {
            $rating = testable_readline("Rating (1-10): ");
        }
        $attributes = [
            new DynamoDBAttribute('rating', 'N', 'HASH', $rating),
            new DynamoDBAttribute('plot', 'S', 'RANGE', $plot)
        ];
        list($statement, $parameters) = $service-
>buildStatementAndParameters("UPDATE", $tableName, $key, $attributes);
        $service->updateItemByPartiQL($statement, $parameters);

        $movie = $service->getItemByPartiQL($tableName, $key);
        echo "Okay, you have rated {$movie['Items'][0]['title']['S']} as a
 {$movie['Items'][0]['rating']['N']}\n";

        $service->deleteItemByPartiQL($statement, $parameters);
        echo "But, bad news, this was a trap. That movie has now been deleted
 because of your rating...harsh.\n";

        echo "That's okay though. The book was better. Now, for something
 lighter, in what year were you born?\n";
        $birthYear = "not a number";
        while (!is_numeric($birthYear) || $birthYear >= date("Y")) {
            $birthYear = testable_readline("Birth year: ");
        }
        $birthKey = [
            'Key' => [
                'year' => [
                    'N' => "$birthYear",
                ],
            ],
        ];
        $result = $service->query($tableName, $birthKey);
        $marshal = new Marshaler();
        echo "Here are the movies in our collection released the year you were
 born:\n";
        $oops = "Oops! There were no movies released in that year (that we know
 of).\n";
```

```
        $display = "";
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            $display .= $movie['title'] . "\n";
        }
        echo ($display) ?: $oops;

        $yearsKey = [
            'Key' => [
                'year' => [
                    'N' => [
                        'minRange' => 1990,
                        'maxRange' => 1999,
                    ],
                ],
            ],
        ];
        $filter = "year between 1990 and 1999";
        echo "\nHere's a list of all the movies released in the 90s:\n";
        $result = $service->scan($tableName, $yearsKey, $filter);
        foreach ($result['Items'] as $movie) {
            $movie = $marshal->unmarshalItem($movie);
            echo $movie['title'] . "\n";
        }

        echo "\nCleaning up this demo by deleting table $tableName...\n";
        $service->deleteTable($tableName);
    }
}

    public function insertItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => "$statement",
            'Parameters' => $parameters,
        ]);
    }

    public function getItemByPartiQL(string $tableName, array $key): Result
    {
        list($statement, $parameters) = $this-
>buildStatementAndParameters("SELECT", $tableName, $key['Item']);

        return $this->dynamoDbClient->executeStatement([
```

```
            'Parameters' => $parameters,
            'Statement' => $statement,
        ]);
    }

    public function updateItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }

    public function deleteItemByPartiQL(string $statement, array $parameters)
    {
        $this->dynamoDbClient->executeStatement([
            'Statement' => $statement,
            'Parameters' => $parameters,
        ]);
    }
```

- Para obtener información sobre la API, consulte ExecuteStatement en la referencia de la API de AWS SDK para PHP.

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Crear una clase que pueda ejecutar instrucciones PartiQL.

```
from datetime import datetime
from decimal import Decimal
import logging
from pprint import pprint
```

```
import boto3
from botocore.exceptions import ClientError

from scaffold import Scaffold

logger = logging.getLogger(__name__)

class PartiQLWrapper:
    """
    Encapsulates a DynamoDB resource to run PartiQL statements.
    """

    def __init__(self, dyn_resource):
        """
        :param dyn_resource: A Boto3 DynamoDB resource.
        """
        self.dyn_resource = dyn_resource


    def run_partiql(self, statement, params):
        """
        Runs a PartiQL statement. A Boto3 resource is used even though
        `execute_statement` is called on the underlying `client` object because
 the
        resource transforms input and output from plain old Python objects
 (POPOs) to
        the DynamoDB format. If you create the client directly, you must do these
        transforms yourself.

        :param statement: The PartiQL statement.
        :param params: The list of PartiQL parameters. These are applied to the
                       statement in the order they are listed.
        :return: The items returned from the statement, if any.
        """
        try:
            output = self.dyn_resource.meta.client.execute_statement(
                Statement=statement, Parameters=params
            )
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "Couldn't execute PartiQL '%s' because the table does not
 exist.",
                    statement,
```

```
                )
            else:
                logger.error(
                    "Couldn't execute PartiQL '%s'. Here's why: %s: %s",
                    statement,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return output
```

Ejecutar un escenario que crea una tabla y ejecuta consultas PartiQL.

```
def run_scenario(scaffold, wrapper, table_name):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon DynamoDB PartiQL single statement demo.")
    print("-" * 88)

    print(f"Creating table '{table_name}' for the demo...")
    scaffold.create_table(table_name)
    print("-" * 88)

    title = "24 Hour PartiQL People"
    year = datetime.now().year
    plot = "A group of data developers discover a new query language they can't
 stop using."
    rating = Decimal("9.9")

    print(f"Inserting movie '{title}' released in {year}.")
    wrapper.run_partiql(
        f"INSERT INTO \"{table_name}\" VALUE {{'title': ?, 'year': ?,
 'info': ?}}",
        [title, year, {"plot": plot, "rating": rating}],
    )
    print("Success!")
    print("-" * 88)
```

```
    print(f"Getting data for movie '{title}' released in {year}.")
    output = wrapper.run_partiql(
        f'SELECT * FROM "{table_name}" WHERE title=? AND year=?', [title, year]
    )
    for item in output["Items"]:
        print(f"\n{item['title']}, {item['year']}")
        pprint(output["Items"])
    print("-" * 88)

    rating = Decimal("2.4")
    print(f"Updating movie '{title}' with a rating of {float(rating)}.")
    wrapper.run_partiql(
        f'UPDATE "{table_name}" SET info.rating=? WHERE title=? AND year=?',
        [rating, title, year],
    )
    print("Success!")
    print("-" * 88)

    print(f"Getting data again to verify our update.")
    output = wrapper.run_partiql(
        f'SELECT * FROM "{table_name}" WHERE title=? AND year=?', [title, year]
    )
    for item in output["Items"]:
        print(f"\n{item['title']}, {item['year']}")
        pprint(output["Items"])
    print("-" * 88)

    print(f"Deleting movie '{title}' released in {year}.")
    wrapper.run_partiql(
        f'DELETE FROM "{table_name}" WHERE title=? AND year=?', [title, year]
    )
    print("Success!")
    print("-" * 88)

    print(f"Deleting table '{table_name}'...")
    scaffold.delete_table()
    print("-" * 88)

    print("\nThanks for watching!")
    print("-" * 88)


if __name__ == "__main__":
    try:
```

```
        dyn_res = boto3.resource("dynamodb")
        scaffold = Scaffold(dyn_res)
        movies = PartiQLWrapper(dyn_res)
        run_scenario(scaffold, movies, "doc-example-table-partiql-movies")
    except Exception as e:
        print(f"Something went wrong with the demo! Here's what: {e}")
```

- Para obtener información sobre la API, consulte ExecuteStatement en Referencia de la API de AWS SDK para Python (Boto3).

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Ejecutar un escenario que crea una tabla y ejecuta consultas PartiQL.

```
table_name = "doc-example-table-movies-partiql-#{rand(10**8)}"
scaffold = Scaffold.new(table_name)
sdk = DynamoDBPartiQLSingle.new(table_name)

new_step(1, 'Create a new DynamoDB table if none already exists.')
unless scaffold.exists?(table_name)
  puts("\nNo such table: #{table_name}. Creating it...")
  scaffold.create_table(table_name)
  print "Done!\n".green
end

new_step(2, 'Populate DynamoDB table with movie data.')
download_file = 'moviedata.json'
puts("Downloading movie database to #{download_file}...")
movie_data = scaffold.fetch_movie_data(download_file)
puts("Writing movie data from #{download_file} into your table...")
scaffold.write_batch(movie_data)
puts("Records added: #{movie_data.length}.")
print "Done!\n".green
```

```
  new_step(3, 'Select a single item from the movies table.')
  response = sdk.select_item_by_title('Star Wars')
  puts("Items selected for title 'Star Wars': #{response.items.length}\n")
  print response.items.first.to_s.yellow
  print "\n\nDone!\n".green

  new_step(4, 'Update a single item from the movies table.')
  puts "Let's correct the rating on The Big Lebowski to 10.0."
  sdk.update_rating_by_title('The Big Lebowski', 1998, 10.0)
  print "\nDone!\n".green

  new_step(5, 'Delete a single item from the movies table.')
  puts "Let's delete The Silence of the Lambs because it's just too scary."
  sdk.delete_item_by_title('The Silence of the Lambs', 1991)
  print "\nDone!\n".green

  new_step(6, 'Insert a new item into the movies table.')
  puts "Let's create a less-scary movie called The Prancing of the Lambs."
  sdk.insert_item('The Prancing of the Lambs', 2005, 'A movie about happy
 livestock.', 5.0)
  print "\nDone!\n".green

  new_step(7, 'Delete the table.')
  return unless scaffold.exists?(table_name)

  scaffold.delete_table
end
```

- Para obtener información sobre la API, consulte ExecuteStatement en la referencia de la API de AWS SDK para Ruby.

Rust

SDK para Rust

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

```
async fn make_table(
    client: &Client,
    table: &str,
    key: &str,
) -> Result<(), SdkError<CreateTableError>> {
    let ad = AttributeDefinition::builder()
        .attribute_name(key)
        .attribute_type(ScalarAttributeType::S)
        .build()
        .expect("creating AttributeDefinition");

    let ks = KeySchemaElement::builder()
        .attribute_name(key)
        .key_type(KeyType::Hash)
        .build()
        .expect("creating KeySchemaElement");

    match client
        .create_table()
        .table_name(table)
        .key_schema(ks)
        .attribute_definitions(ad)
        .billing_mode(BillingMode::PayPerRequest)
        .send()
        .await
    {
        Ok(_) => Ok(()),
        Err(e) => Err(e),
    }
}

async fn add_item(client: &Client, item: Item) -> Result<(),
 SdkError<ExecuteStatementError>> {
    match client
        .execute_statement()
        .statement(format!(
            r#"INSERT INTO "{}" VALUE {{
                "{}": ?,
                "acount_type": ?,
                "age": ?,
                "first_name": ?,
                "last_name": ?
        }} "#,
```

```
            item.table, item.key
        ))
        .set_parameters(Some(vec![
            AttributeValue::S(item.utype),
            AttributeValue::S(item.age),
            AttributeValue::S(item.first_name),
            AttributeValue::S(item.last_name),
        ]))
        .send()
        .await
    {
        Ok(_) => Ok(()),
        Err(e) => Err(e),
    }
}

async fn query_item(client: &Client, item: Item) -> bool {
    match client
        .execute_statement()
        .statement(format!(
            r#"SELECT * FROM "{}" WHERE "{}" = ?"#,
            item.table, item.key
        ))
        .set_parameters(Some(vec![AttributeValue::S(item.value)]))
        .send()
        .await
    {
        Ok(resp) => {
            if !resp.items().is_empty() {
                println!("Found a matching entry in the table:");
                println!("{:?}", resp.items.unwrap_or_default().pop());
                true
            } else {
                println!("Did not find a match.");
                false
            }
        }
        Err(e) => {
            println!("Got an error querying table:");
            println!("{}", e);
            process::exit(1);
        }
    }
}
```

```
async fn remove_item(client: &Client, table: &str, key: &str, value: String) ->
 Result<(), Error> {
    client
        .execute_statement()
        .statement(format!(r#"DELETE FROM "{table}" WHERE "{key}" = ?"#))
        .set_parameters(Some(vec![AttributeValue::S(value)]))
        .send()
        .await?;

    println!("Deleted item.");

    Ok(())
}

async fn remove_table(client: &Client, table: &str) -> Result<(), Error> {
    client.delete_table().table_name(table).send().await?;

    Ok(())
}
```

- Para obtener detalles de la API, consulte ExecuteStatement en la referencia de la API del SDK de AWS para Rust.

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Consulta de una tabla de DynamoDB mediante un índice secundario global con un SDK de AWS

En los siguientes ejemplos de código se muestra cómo consultar una tabla mediante un índice secundario global.

- Consulte una tabla de DynamoDB mediante su clave principal.
- Consulte un índice secundario global (GSI) para patrones de acceso alternativos.
- Compare las consultas de tabla y las consultas de GSI.

Java

SDK para Java 2.x

Consulte una tabla de DynamoDB mediante su clave principal y un índice secundario global (GSI) con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryTable(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();
```

```
       try {
           final QueryResponse response = dynamoDbClient.query(queryRequest);
           System.out.println("Query on base table successful. Found " +
response.count() + " items");
           return response;
       } catch (ResourceNotFoundException e) {
           System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
           throw new DynamoDbQueryException("Table not found: " + tableName, e);
       } catch (DynamoDbException e) {
           System.err.println("Error querying base table: " + e.getMessage());
           throw new DynamoDbQueryException("Failed to execute query on base
table", e);
       }
   }

   /**
    * Queries a DynamoDB Global Secondary Index (GSI) by partition key.
    *
    * @param tableName         The name of the DynamoDB table
    * @param indexName         The name of the GSI
    * @param partitionKeyName  The name of the GSI partition key attribute
    * @param partitionKeyValue The value of the GSI partition key to query
    * @return The query response from DynamoDB
    * @throws ResourceNotFoundException if the table or index doesn't exist
    * @throws DynamoDbException if the query fails
    */
   public QueryResponse queryGlobalSecondaryIndex(
       final String tableName, final String indexName, final String
partitionKeyName, final String partitionKeyValue) {

       CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
       CodeSampleUtils.validateStringParameter("Index name", indexName);

       // Create expression attribute names for the column names
       final Map<String, String> expressionAttributeNames = new HashMap<>();
       expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_IK,
partitionKeyName);

       // Create expression attribute values for the column values
       final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
```

```
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_IK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .indexName(indexName)
            .keyConditionExpression(GSI_KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query on GSI successful. Found " +
response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(
                "Error: The Amazon DynamoDB table \"%s\" or index \"%s\" can't be
found.\n", tableName, indexName);
            throw new DynamoDbQueryException("Table or index not found: " +
tableName + "/" + indexName, e);
        } catch (DynamoDbException e) {
            System.err.println("Error querying GSI: " + e.getMessage());
            throw new DynamoDbQueryException("Failed to execute query on GSI",
e);
        }
    }
```

Compare la consulta directa de una tabla con la consulta de un GSI con AWS SDK for Java 2.x.

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <basePartitionKeyName> <basePartitionKeyValue>
<gsiName> <gsiPartitionKeyName> <gsiPartitionKeyValue> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
```

```
                    basePartitionKeyName - The name of the base table partition
key attribute.
                    basePartitionKeyValue - The value of the base table partition
key to query.
                    gsiName - The name of the Global Secondary Index.
                    gsiPartitionKeyName - The name of the GSI partition key
attribute.
                    gsiPartitionKeyValue - The value of the GSI partition key to
query.
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """";

        if (args.length < 6) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String basePartitionKeyName = args[1];
        final String basePartitionKeyValue = args[2];
        final String gsiName = args[3];
        final String gsiPartitionKeyName = args[4];
        final String gsiPartitionKeyValue = args[5];
        final Region region = args.length > 6 ? Region.of(args[6]) :
Region.US_EAST_1;

        try (DynamoDbClient ddb =
DynamoDbClient.builder().region(region).build()) {
            final QueryTableAndGSI queryHelper = new QueryTableAndGSI(ddb);

            // Query the base table
            System.out.println("Querying base table where " +
basePartitionKeyName + " = " + basePartitionKeyValue);
            final QueryResponse tableResponse =
                queryHelper.queryTable(tableName, basePartitionKeyName,
basePartitionKeyValue);

            System.out.println("Found " + tableResponse.count() + " items in base
table:");
            tableResponse.items().forEach(item -> System.out.println(item));

            // Query the GSI
            System.out.println(
```

```
                "\nQuerying GSI '" + gsiName + "' where " + gsiPartitionKeyName +
" = " + gsiPartitionKeyValue);
            final QueryResponse gsiResponse =
                queryHelper.queryGlobalSecondaryIndex(tableName, gsiName,
gsiPartitionKeyName, gsiPartitionKeyValue);

            System.out.println("Found " + gsiResponse.count() + " items in
GSI:");
            gsiResponse.items().forEach(item -> System.out.println(item));

            // Explain the differences between querying a table and a GSI
            System.out.println("\nKey differences between querying a table and a
GSI:");
            System.out.println("1. When querying a GSI, you must specify the
indexName parameter");
            System.out.println("2. GSIs may not contain all attributes from the
base table (projection)");
            System.out.println("3. GSIs consume read capacity units from the
GSI's capacity, not the base table's");
            System.out.println("4. GSIs may have eventually consistent data
(cannot use ConsistentRead=true)");

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table or index not found: " + e.getMessage());
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte una tabla de DynamoDB mediante la clave principal con AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
  config,
  tableName,
  userId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
    const input = {
      TableName: tableName,
      KeyConditionExpression: "user_id = :userId",
      ExpressionAttributeValues: {
        ":userId": { S: userId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying table: ${error}`);
    throw error;
  }
}
```

Consulte un índice secundario global (GSI) de DynamoDB con AWS SDK para JavaScript.

```
/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
  config,
  tableName,
  indexName,
  gameId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the GSI
    const input = {
      TableName: tableName,
      IndexName: indexName,
      KeyConditionExpression: "game_id = :gameId",
      ExpressionAttributeValues: {
        ":gameId": { S: gameId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying GSI: ${error}`);
    throw error;
  }
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte una tabla de DynamoDB mediante su clave principal y un índice secundario global (GSI) con AWS SDK for Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Key


def query_table(table_name, partition_key_name, partition_key_value):
    """
    Query a DynamoDB table using its primary key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query on the table's primary key
    response =
 table.query(KeyConditionExpression=Key(partition_key_name).eq(partition_key_value))

    return response


def query_gsi(table_name, index_name, partition_key_name, partition_key_value):
    """
    Query a Global Secondary Index (GSI) on a DynamoDB table.

    Args:
        table_name (str): The name of the DynamoDB table.
        index_name (str): The name of the Global Secondary Index.
        partition_key_name (str): The name of the GSI's partition key attribute.
        partition_key_value (str): The value of the GSI's partition key to query.
```

```
    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query on the GSI
    response = table.query(
        IndexName=index_name,
KeyConditionExpression=Key(partition_key_name).eq(partition_key_value)
    )

    return response
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Consulta de una tabla de DynamoDB mediante una condición begins_with con un SDK de AWS

En los siguientes ejemplos de código se muestra cómo consultar una tabla mediante una condición begins_with.

- Utilice la función begins_with en una expresión de condición de clave.
- Filtre los elementos según un patrón de prefijos en la clave de clasificación.

Java

SDK para Java 2.x

Consulte una tabla de DynamoDB mediante una condición begins_with en la clave de clasificación con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithBeginsWithCondition(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String sortKeyName,
        final String sortKeyPrefix) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Sort key name", sortKeyName);
        CodeSampleUtils.validateStringParameter("Sort key prefix",
 sortKeyPrefix);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_SK, sortKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_SK_PREFIX,
            AttributeValue.builder().s(sortKeyPrefix).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
```

```
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query with begins_with condition successful.
Found {0} items", response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with begins_with condition",
e);
            throw e;
        }
    }
```

Demuestre el uso de begins_with con prefijos de diferente longitud con AWS SDK for Java 2.x.

```
    public static void main(String[] args) {
        try {
            CodeSampleUtils.BeginsWithQueryConfig config =
CodeSampleUtils.BeginsWithQueryConfig.fromArgs(args);
            LOGGER.log(Level.INFO, "Querying items where {0} = {1} and {2} begins
with ''{3}''", new Object[] {
                config.getPartitionKeyName(),
                config.getPartitionKeyValue(),
                config.getSortKeyName(),
                config.getSortKeyPrefix()
            });

            // Using the builder pattern to create and execute the query
            final QueryResponse response = new BeginsWithQueryBuilder()
                .withTableName(config.getTableName())
                .withPartitionKeyName(config.getPartitionKeyName())
                .withPartitionKeyValue(config.getPartitionKeyValue())
                .withSortKeyName(config.getSortKeyName())
                .withSortKeyPrefix(config.getSortKeyPrefix())
```

```
                .withRegion(config.getRegion())
                .execute();

            // Process the results
            LOGGER.log(Level.INFO, "Found {0} items:", response.count());
            response.items().forEach(item -> LOGGER.info(item.toString()));

            // Demonstrate with a different prefix
            if (!config.getSortKeyPrefix().isEmpty()) {
                String shorterPrefix = config.getSortKeyPrefix()
                    .substring(0, Math.max(1,
config.getSortKeyPrefix().length() / 2));
                LOGGER.log(Level.INFO, "\nNow querying with a shorter prefix:
''{0}''", shorterPrefix);

                final QueryResponse response2 = new BeginsWithQueryBuilder()
                    .withTableName(config.getTableName())
                    .withPartitionKeyName(config.getPartitionKeyName())
                    .withPartitionKeyValue(config.getPartitionKeyValue())
                    .withSortKeyName(config.getSortKeyName())
                    .withSortKeyPrefix(shorterPrefix)
                    .withRegion(config.getRegion())
                    .execute();

                LOGGER.log(Level.INFO, "Found {0} items with shorter prefix:",
response2.count());
                response2.items().forEach(item -> LOGGER.info(item.toString()));
            }
        } catch (IllegalArgumentException e) {
            LOGGER.log(Level.SEVERE, "Invalid input: {0}", e.getMessage());
            printUsage();
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found", e);
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "DynamoDB error", e);
        } catch (Exception e) {
            LOGGER.log(Level.SEVERE, "Unexpected error", e);
        }
    }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte una tabla de DynamoDB mediante una condición begins_with en la clave de clasificación con AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items where the sort key begins with a specific
 prefix
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key
 * @param {string} prefix - The prefix to match at the beginning of the sort key
 * @returns {Promise<Object>} - The query response
 */
async function queryWithBeginsWith(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  prefix
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND begins_with(#sk, :prefix)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":prefix": { S: prefix }
```

```
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with begins_with: ${error}`);
    throw error;
  }
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte una tabla de DynamoDB mediante una condición begins_with en la clave de clasificación con AWS SDK for Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Key


def query_with_begins_with(
    table_name, partition_key_name, partition_key_value, sort_key_name, prefix
):
    """
    Query a DynamoDB table with a begins_with condition on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute.
        prefix (str): The prefix to match at the beginning of the sort key.

    Returns:
        dict: The response from DynamoDB containing the query results.
```

```
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Perform the query with a begins_with condition on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) & Key(
        sort_key_name
    ).begins_with(prefix)
    response = table.query(KeyConditionExpression=key_condition)

    return response
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Consulta de una tabla de DynamoDB mediante un intervalo de fechas en la clave de clasificación con un SDK de AWS

En los siguientes ejemplos de código se muestra cómo consultar una tabla con un intervalo de fechas en la clave de clasificación.

- Consulte los elementos dentro de un intervalo de fechas específico.
- Utilice operadores de comparación en las claves de clasificación con formato de fecha.

Java

SDK para Java 2.x

Consulte una tabla de DynamoDB para buscar elementos dentro de un intervalo de fechas con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.LocalDate;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithDateRange(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final LocalDate startDate,
        final LocalDate endDate) {

        // Focus on query logic, assuming parameters are valid
        if (startDate == null || endDate == null) {
            throw new IllegalArgumentException("Start date and end date cannot be
 null");
        }

        if (endDate.isBefore(startDate)) {
            throw new IllegalArgumentException("End date must be after start
 date");
        }

        // Format dates as ISO strings for DynamoDB (using just the date part)
        final String formattedStartDate = startDate.toString();
        final String formattedEndDate = endDate.toString();

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_SK, dateKeyName);

        // Create expression attribute values for the column values
```

```
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_START_DATE,
            AttributeValue.builder().s(formattedStartDate).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_END_DATE,
            AttributeValue.builder().s(formattedEndDate).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query by date range successful. Found {0}
items", response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying by date range: {0}",
e.getMessage());
            throw e;
        }
    }
```

Muestra cómo consultar una tabla de DynamoDB con filtrado por intervalo de fechas.

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
```

```
                    <tableName> <partitionKeyName> <partitionKeyValue>
<dateKeyName> <startDate> <endDate> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    dateKeyName - The name of the date attribute to filter on.
                    startDate - The start date for the range query (YYYY-MM-DD).
                    endDate - The end date for the range query (YYYY-MM-DD).
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 6) {
            System.out.println(usage);
            System.exit(1);
        }

        try {
            // Parse command line arguments into a config object
            CodeSampleUtils.DateRangeQueryConfig config =
CodeSampleUtils.DateRangeQueryConfig.fromArgs(args);

            LOGGER.log(
                Level.INFO, "Querying items from {0} to {1}", new Object[]
{config.getStartDate(), config.getEndDate()
                });

            // Using the builder pattern to create and execute the query
            final QueryResponse response = new DateRangeQueryBuilder()
                .withTableName(config.getTableName())
                .withPartitionKeyName(config.getPartitionKeyName())
                .withPartitionKeyValue(config.getPartitionKeyValue())
                .withDateKeyName(config.getDateKeyName())
                .withStartDate(config.getStartDate())
                .withEndDate(config.getEndDate())
                .withRegion(config.getRegion())
                .execute();

            // Process the results
            LOGGER.log(Level.INFO, "Found {0} items:", response.count());
            response.items().forEach(item -> {
                LOGGER.info(item.toString());
```

```
                // Extract and display the date attribute for clarity
                if (item.containsKey(config.getDateKeyName())) {
                    LOGGER.log(
                        Level.INFO,
                        "  Date attribute: {0}",
                        item.get(config.getDateKeyName()).s());
                }
            });

            // Demonstrate with a different date range
            LocalDate narrowerStartDate = config.getStartDate().plusDays(1);
            LocalDate narrowerEndDate = config.getEndDate().minusDays(1);

            if (!narrowerStartDate.isAfter(narrowerEndDate)) {
                LOGGER.log(Level.INFO, "\nNow querying with a narrower date
range: {0} to {1}", new Object[] {
                    narrowerStartDate, narrowerEndDate
                });

                final QueryResponse response2 = new DateRangeQueryBuilder()
                    .withTableName(config.getTableName())
                    .withPartitionKeyName(config.getPartitionKeyName())
                    .withPartitionKeyValue(config.getPartitionKeyValue())
                    .withDateKeyName(config.getDateKeyName())
                    .withStartDate(narrowerStartDate)
                    .withEndDate(narrowerEndDate)
                    .withRegion(config.getRegion())
                    .execute();

                LOGGER.log(Level.INFO, "Found {0} items with narrower date
range:", response2.count());
                response2.items().forEach(item -> LOGGER.info(item.toString()));
            }

            LOGGER.info("\nNote: When storing dates in DynamoDB:");
            LOGGER.info("1. Use ISO format (YYYY-MM-DD) for lexicographical
ordering");
            LOGGER.info("2. Use the BETWEEN operator for inclusive date range
queries");
            LOGGER.info("3. Consider using ISO-8601 format for timestamps with
time components");

        } catch (IllegalArgumentException e) {
            LOGGER.log(Level.SEVERE, "Invalid input: {0}", e.getMessage());
```

```
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", e.getMessage());
            System.exit(1);
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "DynamoDB error: {0}", e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            LOGGER.log(Level.SEVERE, "Unexpected error: {0}", e.getMessage());
            System.exit(1);
        }
    }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte una tabla de DynamoDB para buscar elementos dentro de un intervalo de fechas con AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range on the sort
 key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
 attribute)
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRangeOnSortKey(
  config,
```

```
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
 AND :endDate',
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range on sort key: ${error}`);
    throw error;
  }
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte una tabla de DynamoDB para buscar elementos dentro de un intervalo de fechas con AWS SDK for Python (Boto3).

```
from datetime import datetime, timedelta

import boto3
from boto3.dynamodb.conditions import Key


def query_with_date_range(
    table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
):
    """
    Query a DynamoDB table with a date range on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        start_date (datetime): The start date for the query range.
        end_date (datetime): The end date for the query range.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Format the date values as ISO 8601 strings
    # DynamoDB works well with ISO format for date values
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key using BETWEEN operator
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
```

```
        start_date_str, end_date_str
    )

    response = table.query(
        KeyConditionExpression=key_condition,
        ExpressionAttributeValues={
            ":pk_val": partition_key_value,
            ":start_date": start_date_str,
            ":end_date": end_date_str,
        },
    )

    return response


def query_with_date_range_by_month(
    table_name, partition_key_name, partition_key_value, sort_key_name, year,
 month
):
    """
    Query a DynamoDB table for a specific month's data.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        year (int): The year to query.
        month (int): The month to query (1-12).

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Calculate the start and end dates for the specified month
    if month == 12:
        next_year = year + 1
        next_month = 1
    else:
        next_year = year
```

```
        next_month = month + 1

    start_date = datetime(year, month, 1)
    end_date = datetime(next_year, next_month, 1) - timedelta(microseconds=1)

    # Format the date values as ISO 8601 strings
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) &
Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(KeyConditionExpression=key_condition)

    return response
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Consulta de una tabla de DynamoDB con una expresión de filtro compleja con un SDK de AWS

En los siguientes ejemplos de código se muestra cómo consultar una tabla con una expresión de filtro compleja.

- Aplique expresiones de filtro complejas a los resultados de la consulta.
- Combine condiciones múltiples mediante el uso de operadores lógicos.
- Filtre los elementos en función de atributos sin clave.

Java

SDK para Java 2.x

Consulte una tabla de DynamoDB con una expresión de filtro compleja mediante AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithComplexFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String statusAttrName,
        final String activeStatus,
        final String pendingStatus,
        final String priceAttrName,
        final double minPrice,
        final double maxPrice,
        final String categoryAttrName) {

        // Validate parameters
        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
       CodeSampleUtils.validateStringParameter("Status attribute name",
 statusAttrName);
       CodeSampleUtils.validateStringParameter("Active status", activeStatus);
       CodeSampleUtils.validateStringParameter("Pending status", pendingStatus);
       CodeSampleUtils.validateStringParameter("Price attribute name",
 priceAttrName);
       CodeSampleUtils.validateStringParameter("Category attribute name",
 categoryAttrName);
```

```
        CodeSampleUtils.validateNumericRange("Minimum price", minPrice, 0.0,
Double.MAX_VALUE);
        CodeSampleUtils.validateNumericRange("Maximum price", maxPrice, minPrice,
Double.MAX_VALUE);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pk", partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_STATUS,
statusAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PRICE,
priceAttrName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_CATEGORY,
categoryAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            ":pkValue", AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_ACTIVE,
            AttributeValue.builder().s(activeStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PENDING,
            AttributeValue.builder().s(pendingStatus).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MIN_PRICE,
            AttributeValue.builder().n(String.valueOf(minPrice)).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_MAX_PRICE,
            AttributeValue.builder().n(String.valueOf(maxPrice)).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        return dynamoDbClient.query(queryRequest);
    }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte una tabla de DynamoDB con una expresión de filtro compleja mediante AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with a complex filter expression
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number|string} minViews - Minimum number of views for filtering
 * @param {number|string} minReplies - Minimum number of replies for filtering
 * @param {string} requiredTag - Tag that must be present in the item's tags set
 * @returns {Promise<Object>} - The query response
 */
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
```

```
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
 contains(tags, :tag)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with complex filter: ${error}`);
    throw error;
  }
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte una tabla de DynamoDB con una expresión de filtro compleja mediante AWS SDK for Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_complex_filter(
    table_name,
    partition_key_name,
    partition_key_value,
```

```
    min_rating=None,
    status_list=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        min_rating (float, optional): Minimum rating value for filtering.
        status_list (list, optional): List of status values to include.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize the filter expression and expression attribute values
    filter_expression = None
    expression_attribute_values = {}

    # Build the filter expression based on provided parameters
    if min_rating is not None:
        filter_expression = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if status_list and len(status_list) > 0:
        status_condition = None
        for i, status in enumerate(status_list):
            status_value_name = f":status{i}"
            expression_attribute_values[status_value_name] = status

            if status_condition is None:
                status_condition = Attr("status").eq(status)
            else:
                status_condition = status_condition | Attr("status").eq(status)
```

```
        if filter_expression is None:
            filter_expression = status_condition
        else:
            filter_expression = filter_expression & status_condition

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if filter_expression is None:
            filter_expression = price_condition
        else:
            filter_expression = filter_expression & price_condition

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
 expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response


def query_with_complex_filter_and_or(
    table_name,
    partition_key_name,
    partition_key_value,
    category=None,
    min_rating=None,
    max_price=None,
):
    """
    Query a DynamoDB table with a complex filter expression using AND and OR
 operators.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
```

```
        partition_key_value (str): The value of the partition key to query.
        category (str, optional): Category value for filtering.
        min_rating (float, optional): Minimum rating value for filtering.
        max_price (float, optional): Maximum price value for filtering.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Build a complex filter expression with AND and OR operators
    filter_expression = None
    expression_attribute_values = {}

    # Build the category condition
    if category:
        filter_expression = Attr("category").eq(category)
        expression_attribute_values[":category"] = category

    # Build the rating and price condition (rating >= min_rating OR price <=
max_price)
    rating_price_condition = None

    if min_rating is not None:
        rating_price_condition = Attr("rating").gte(min_rating)
        expression_attribute_values[":min_rating"] = min_rating

    if max_price is not None:
        price_condition = Attr("price").lte(max_price)
        expression_attribute_values[":max_price"] = max_price

        if rating_price_condition is None:
            rating_price_condition = price_condition
        else:
            rating_price_condition = rating_price_condition | price_condition

    # Combine the conditions
    if rating_price_condition:
        if filter_expression is None:
```

```
            filter_expression = rating_price_condition
        else:
            filter_expression = filter_expression & rating_price_condition

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression
        if expression_attribute_values:
            query_params["ExpressionAttributeValues"] =
expression_attribute_values

    # Execute the query
    response = table.query(**query_params)
    return response
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Consulta de una tabla de DynamoDB con una expresión de filtro compleja con un SDK de AWS

En los siguientes ejemplos de código se muestra cómo consultar una tabla con una expresión de filtro dinámica.

- Cree expresiones de filtro de forma dinámica en tiempo de ejecución.
- Cree condiciones de filtro en función de las entradas del usuario o del estado de la aplicación.
- Agregue o elimine criterios de filtro de forma condicional.

Java

SDK para Java 2.x

Consulte una tabla de DynamoDB con una expresión de filtro construida dinámicamente mediante AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public static QueryResponse queryWithDynamicFilter(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final Map<String, Object> filterCriteria,
        final Region region,
        final DynamoDbClient dynamoDbClient) {

        validateParameters(tableName, partitionKeyName, partitionKeyValue,
 filterCriteria);

        DynamoDbClient ddbClient = dynamoDbClient;
        boolean shouldClose = false;

        try {
            if (ddbClient == null) {
                ddbClient = createClient(region);
                shouldClose = true;
            }

            final QueryWithDynamicFilter queryHelper = new
 QueryWithDynamicFilter(ddbClient);
            return queryHelper.queryWithDynamicFilter(tableName,
 partitionKeyName, partitionKeyValue, filterCriteria);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
```

```
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Failed to execute dynamic filter query: " +
e.getMessage());
            throw e;
        } catch (Exception e) {
            System.err.println("Unexpected error during query: " +
e.getMessage());
            throw e;
        } finally {
            if (shouldClose && ddbClient != null) {
                ddbClient.close();
            }
        }
    }
```

Demuestra cómo utilizar expresiones de filtro dinámico con AWS SDK for Java 2.x.

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue>
<filterAttrName> <filterAttrValue> [region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    filterAttrName - The name of the attribute to filter on.
                    filterAttrValue - The value to filter by.
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 5) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String partitionKeyName = args[1];
        final String partitionKeyValue = args[2];
```

```
        final String filterAttrName = args[3];
        final String filterAttrValue = args[4];
        final Region region = args.length > 5 ? Region.of(args[5]) :
Region.US_EAST_1;

        System.out.println("Querying items with dynamic filter: " +
filterAttrName + " = " + filterAttrValue);

        try {
            // Using the builder pattern to create and execute the query
            final QueryResponse response = new DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriterion(filterAttrName, filterAttrValue)
                .withRegion(region)
                .execute();

            // Process the results
            System.out.println("Found " + response.count() + " items:");
            response.items().forEach(item -> System.out.println(item));

            // Demonstrate multiple filter criteria
            System.out.println("\nNow querying with multiple filter criteria:");

            Map<String, Object> multipleFilters = new HashMap<>();
            multipleFilters.put(filterAttrName, filterAttrValue);
            multipleFilters.put("status", "active");

            final QueryResponse multiFilterResponse = new
DynamicFilterQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withFilterCriteria(multipleFilters)
                .withRegion(region)
                .execute();

            System.out.println("Found " + multiFilterResponse.count() + " items
with multiple filters:");
            multiFilterResponse.items().forEach(item ->
System.out.println(item));

        } catch (IllegalArgumentException e) {
```

```
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte una tabla de DynamoDB con una expresión de filtro construida dinámicamente mediante AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

async function queryWithDynamicFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
```

```
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
      ":skValue": { S: sortKeyValue }
    };
    const expressionAttributeNames = {
      "#pk": partitionKeyName,
      "#sk": sortKeyName
    };

    // Add status filter if provided
    if (filterParams.status) {
      filterExpressions.push("status = :status");
      expressionAttributeValues[":status"] = { S: filterParams.status };
    }

    // Add minimum views filter if provided
    if (filterParams.minViews !== undefined) {
      filterExpressions.push("views >= :minViews");
      expressionAttributeValues[":minViews"] = { N:
filterParams.minViews.toString() };
    }

    // Add author filter if provided
    if (filterParams.author) {
      filterExpressions.push("author = :author");
      expressionAttributeValues[":author"] = { S: filterParams.author };
    }

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
    };

    // Add filter expression if any filters were provided
    if (filterExpressions.length > 0) {
      input.FilterExpression = filterExpressions.join(" AND ");
    }

    // Add expression attribute names and values
    input.ExpressionAttributeNames = expressionAttributeNames;
    input.ExpressionAttributeValues = expressionAttributeValues;

    // Execute the query
```

```
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with dynamic filter: ${error}`);
    throw error;
  }
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte una tabla de DynamoDB con una expresión de filtro construida dinámicamente mediante AWS SDK for Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_dynamic_filter(
    table_name, partition_key_name, partition_key_value, filter_conditions=None
):
    """
    Query a DynamoDB table with a dynamically constructed filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_conditions (dict, optional): A dictionary of filter conditions
 where
            keys are attribute names and values are dictionaries with 'operator'
 and 'value'.
            Example: {'rating': {'operator': '>=', 'value': 4}, 'status':
 {'operator': '=', 'value': 'active'}}

    Returns:
        dict: The response from DynamoDB containing the query results.
```

```
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Start with the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Initialize variables for the filter expression and attribute values
    filter_expression = None
    expression_attribute_values = {":pk_val": partition_key_value}

    # Dynamically build the filter expression if filter conditions are provided
    if filter_conditions:
        for attr_name, condition in filter_conditions.items():
            operator = condition.get("operator")
            value = condition.get("value")
            attr_value_name = f":{attr_name}"
            expression_attribute_values[attr_value_name] = value

            # Create the appropriate filter expression based on the operator
            current_condition = None
            if operator == "=":
                current_condition = Attr(attr_name).eq(value)
            elif operator == "!=":
                current_condition = Attr(attr_name).ne(value)
            elif operator == ">":
                current_condition = Attr(attr_name).gt(value)
            elif operator == ">=":
                current_condition = Attr(attr_name).gte(value)
            elif operator == "<":
                current_condition = Attr(attr_name).lt(value)
            elif operator == "<=":
                current_condition = Attr(attr_name).lte(value)
            elif operator == "contains":
                current_condition = Attr(attr_name).contains(value)
            elif operator == "begins_with":
                current_condition = Attr(attr_name).begins_with(value)

            # Combine with existing filter expression using AND
            if current_condition:
                if filter_expression is None:
                    filter_expression = current_condition
                else:
```

```
                    filter_expression = filter_expression & current_condition

    # Perform the query with the dynamically built filter expression
    query_params = {"KeyConditionExpression": key_condition}

    if filter_expression:
        query_params["FilterExpression"] = filter_expression

    response = table.query(**query_params)
    return response
```

Demuestra cómo utilizar expresiones de filtro dinámico con AWS SDK for Python (Boto3).

```
def example_usage():
    """Example of how to use the query_with_dynamic_filter function."""
    # Example parameters
    table_name = "Products"
    partition_key_name = "Category"
    partition_key_value = "Electronics"

    # Define dynamic filter conditions based on user input or runtime conditions
    user_min_rating = 4  # This could come from user input
    user_status_filter = "active"  # This could come from user input

    filter_conditions = {}

    # Only add conditions that are actually specified
    if user_min_rating is not None:
        filter_conditions["rating"] = {"operator": ">=", "value":
 user_min_rating}

    if user_status_filter:
        filter_conditions["status"] = {"operator": "=", "value":
 user_status_filter}

    print(
        f"Querying products in category '{partition_key_value}' with filter
 conditions: {filter_conditions}"
    )
```

```
    # Execute the query with dynamic filter
    response = query_with_dynamic_filter(
        table_name, partition_key_name, partition_key_value, filter_conditions
    )

    # Process the results
    items = response.get("Items", [])
    print(f"Found {len(items)} items")

    for item in items:
        print(f"Product: {item}")
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Consulta de una tabla de DynamoDB con una expresión de filtro y límite con un SDK de AWS

En los siguientes ejemplos de código se muestra cómo consultar una tabla con una expresión de filtro y límite.

- Aplique expresiones de filtro a los resultados de las consultas con un límite de elementos evaluados.
- Comprenda cómo afecta el límite a los resultados de las consultas filtradas.
- Control de la cantidad máxima de elementos que se procesan en una consulta.

Java

SDK para Java 2.x

Consulte una tabla de DynamoDB con una expresión de filtro y límite mediante AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithFilterAndLimit(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String filterAttrName,
        final String filterAttrValue,
        final int limit) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Filter attribute name",
 filterAttrName);
        CodeSampleUtils.validateStringParameter("Filter attribute value",
 filterAttrValue);
        CodeSampleUtils.validatePositiveInteger("Limit", limit);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_FILTER,
 filterAttrName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
```

```
            EXPRESSION_ATTRIBUTE_VALUE_FILTER,
            AttributeValue.builder().s(filterAttrValue).build());

        // Create the filter expression
        final String filterExpression = "#filterAttr = :filterValue";

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(filterExpression)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .limit(limit)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query with filter and limit successful. Found
{0} items", response.count());
            LOGGER.log(
                Level.INFO, "ScannedCount: {0} (total items evaluated before
filtering)", response.scannedCount());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with filter and limit: {0}",
e.getMessage());
            throw e;
        }
    }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK for Java 2.x.

Python

SDK para Python (Boto3)

Consulte una tabla de DynamoDB con una expresión de filtro y límite mediante AWS SDK for Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_filter_and_limit(
    table_name,
    partition_key_name,
    partition_key_value,
    filter_attribute=None,
    filter_value=None,
    limit=10,
):
    """
    Query a DynamoDB table with a filter expression and limit the number of
 results.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_attribute (str, optional): The attribute name to filter on.
        filter_value (any, optional): The value to compare against in the filter.
        limit (int, optional): The maximum number of items to evaluate. Defaults
 to 10.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Prepare the query parameters
    query_params = {"KeyConditionExpression": key_condition, "Limit": limit}
```

```
    # Add the filter expression if filter attributes are provided
    if filter_attribute and filter_value is not None:
        query_params["FilterExpression"] =
 Attr(filter_attribute).gt(filter_value)
        query_params["ExpressionAttributeValues"] = {":filter_value":
 filter_value}

    # Execute the query
    response = table.query(**query_params)
    return response
```

Demuestra cómo utilizar expresiones de filtro con límites en AWS SDK for Python (Boto3).

```
def example_usage():
    """Example of how to use the query_with_filter_and_limit function."""
    # Example parameters
    table_name = "ProductReviews"
    partition_key_name = "ProductId"
    partition_key_value = "P123456"
    filter_attribute = "Rating"
    filter_value = 3  # Filter for ratings > 3
    limit = 5

    print(f"Querying reviews for product '{partition_key_value}' with rating >
 {filter_value}")
    print(f"Limiting to {limit} evaluated items")

    # Execute the query with filter and limit
    response = query_with_filter_and_limit(
        table_name, partition_key_name, partition_key_value, filter_attribute,
 filter_value, limit
    )

    # Process the results
    items = response.get("Items", [])
    print(f"\nReturned {len(items)} items that passed the filter")

    for item in items:
        print(f"Review: {item}")
```

```
    # Explain the difference between Limit and actual results
    explain_limit_vs_results(response)

    # Check if there are more results
    if "LastEvaluatedKey" in response:
        print("\nThere are more results available. Use the LastEvaluatedKey for
pagination.")
    else:
        print("\nAll matching results have been retrieved.")
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Consulta de una tabla de DynamoDB con atributos anidados mediante un SDK de AWS

En los siguientes ejemplos de código se muestra cómo consultar una tabla con atributos anidados.

- Acceda y filtre por atributos anidados en los elementos de DynamoDB.
- Utilice expresiones de ruta de documentos para hacer referencia a elementos anidados.

Java

SDK para Java 2.x

Consulte una tabla de DynamoDB con atributos anidados mediante AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
```

```
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryWithNestedAttributes(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String nestedPath,
        final String nestedAttr,
        final String nestedValue) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Nested path", nestedPath);
        CodeSampleUtils.validateStringParameter("Nested attribute", nestedAttr);
        CodeSampleUtils.validateStringParameter("Nested value", nestedValue);

        // Split the nested path into components
        final String[] pathComponents = nestedPath.split("\\.");

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Build the nested attribute reference using document path notation
        final StringBuilder nestedAttributeRef = new StringBuilder();
        for (int i = 0; i < pathComponents.length; i++) {
            final String aliasName = "#n" + i;
            expressionAttributeNames.put(aliasName, pathComponents[i]);

            if (i > 0) {
                nestedAttributeRef.append(".");
            }
            nestedAttributeRef.append(aliasName);
        }

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
```

```
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_NESTED,
            AttributeValue.builder().s(nestedValue).build());

        // Create the filter expression using the nested attribute reference
        final String filterExpression = nestedAttributeRef + " = :nestedValue";

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(filterExpression)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query with nested attribute filter successful.
Found " + response.count() + " items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with nested attribute filter: " +
e.getMessage());
            throw e;
        }
    }
```

Muestra cómo consultar una tabla de DynamoDB con atributos anidados.

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue>
<nestedPath> <nestedAttr> <nestedValue> [region]
```

```
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    nestedPath - The path to the nested map attribute (e.g.,
"address").
                    nestedAttr - The name of the nested attribute (e.g., "city").
                    nestedValue - The value to filter by (e.g., "Seattle").
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 6) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String partitionKeyName = args[1];
        final String partitionKeyValue = args[2];
        final String nestedPath = args[3];
        final String nestedAttr = args[4];
        final String nestedValue = args[5];
        final Region region = args.length > 6 ? Region.of(args[6]) :
Region.US_EAST_1;

        System.out.println("Querying items where " + partitionKeyName + " = " +
partitionKeyValue + " and " + nestedPath
            + "." + nestedAttr + " = " + nestedValue);

        try {
            // Using the builder pattern to create and execute the query
            final QueryResponse response = new NestedAttributeQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withNestedPath(nestedPath)
                .withNestedAttribute(nestedAttr)
                .withNestedValue(nestedValue)
                .withRegion(region)
                .execute();

            // Process the results
            System.out.println("Found " + response.count() + " items:");
```

```
            response.items().forEach(item -> {
                System.out.println(item);

                // Extract and display the nested attribute for clarity
                if (item.containsKey(nestedPath) && item.get(nestedPath).hasM())
{
                    Map<String, AttributeValue> nestedMap =
item.get(nestedPath).m();
                    if (nestedMap.containsKey(nestedAttr)) {
                        System.out.println("  Nested attribute " + nestedPath +
"." + nestedAttr + ": "
                                + formatAttributeValue(nestedMap.get(nestedAttr)));
                    }
                }
            });

            System.out.println("\nNote: When working with nested attributes in
DynamoDB:");
            System.out.println("1. Use dot notation in filter expressions to
access nested attributes");
            System.out.println("2. Use expression attribute names for each
component of the path");
            System.out.println("3. Check if the nested attribute exists before
accessing it");

        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
        }
    }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte una tabla de DynamoDB con atributos anidados mediante AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table filtering on a nested attribute
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} productId - The product ID to query by (partition key)
 * @param {string} category - The category to filter by (nested attribute)
 * @returns {Promise<Object>} - The query response
 */
async function queryWithNestedAttribute(
  config,
  tableName,
  productId,
  category
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "product_id = :productId",
      FilterExpression: "details.category = :category",
      ExpressionAttributeValues: {
        ":productId": { S: productId },
        ":category": { S: category }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with nested attribute: ${error}`);
```

```
    throw error;
  }
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte una tabla de DynamoDB con atributos anidados mediante AWS SDK for Python (Boto3).

```
from typing import Any, Dict, List

import boto3
from boto3.dynamodb.conditions import Attr, Key


def query_with_nested_attributes(
    table_name: str,
    partition_key_name: str,
    partition_key_value: str,
    nested_path: str,
    comparison_operator: str,
    comparison_value: Any,
) -> Dict[str, Any]:
    """
    Query a DynamoDB table and filter by nested attributes.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        nested_path (str): The path to the nested attribute (e.g.,
 'specs.weight').
        comparison_operator (str): The comparison operator to use ('=', '!=',
 '<', '<=', '>', '>=').
        comparison_value (any): The value to compare against.
```

```
    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Build the filter expression based on the nested attribute path and
 comparison operator
    filter_expression = None
    if comparison_operator == "=":
        filter_expression = Attr(nested_path).eq(comparison_value)
    elif comparison_operator == "!=":
        filter_expression = Attr(nested_path).ne(comparison_value)
    elif comparison_operator == "<":
        filter_expression = Attr(nested_path).lt(comparison_value)
    elif comparison_operator == "<=":
        filter_expression = Attr(nested_path).lte(comparison_value)
    elif comparison_operator == ">":
        filter_expression = Attr(nested_path).gt(comparison_value)
    elif comparison_operator == ">=":
        filter_expression = Attr(nested_path).gte(comparison_value)
    elif comparison_operator == "contains":
        filter_expression = Attr(nested_path).contains(comparison_value)
    elif comparison_operator == "begins_with":
        filter_expression = Attr(nested_path).begins_with(comparison_value)

    # Execute the query with the filter expression
    response = table.query(KeyConditionExpression=key_condition,
 FilterExpression=filter_expression)

    return response


def query_with_multiple_nested_attributes(
    table_name: str,
    partition_key_name: str,
    partition_key_value: str,
    nested_conditions: List[Dict[str, Any]],
) -> Dict[str, Any]:
    """
```

```
    Query a DynamoDB table and filter by multiple nested attributes.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        nested_conditions (list): A list of dictionaries, each containing:
            - path (str): The path to the nested attribute
            - operator (str): The comparison operator
            - value (any): The value to compare against

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    # Build the combined filter expression for all nested attributes
    combined_filter = None

    for condition in nested_conditions:
        if not isinstance(condition, dict):
            continue
        path = condition.get("path", "")
        operator = condition.get("operator", "")
        value = condition.get("value")

        if not path or not operator:
            continue

        # Build the individual filter expression
        current_filter = None
        if operator == "=":
            current_filter = Attr(path).eq(value)
        elif operator == "!=":
            current_filter = Attr(path).ne(value)
        elif operator == "<":
            current_filter = Attr(path).lt(value)
        elif operator == "<=":
            current_filter = Attr(path).lte(value)
```

```
        elif operator == ">":
            current_filter = Attr(path).gt(value)
        elif operator == ">=":
            current_filter = Attr(path).gte(value)
        elif operator == "contains":
            current_filter = Attr(path).contains(value)
        elif operator == "begins_with":
            current_filter = Attr(path).begins_with(value)

        # Combine with the existing filter using AND
        if current_filter:
            if combined_filter is None:
                combined_filter = current_filter
            else:
                combined_filter = combined_filter & current_filter

    # Execute the query with the combined filter expression
    response = table.query(KeyConditionExpression=key_condition,
FilterExpression=combined_filter)

    return response
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Consulta de una tabla de DynamoDB con paginación mediante un SDK de AWS

En los siguientes ejemplos de código se muestra cómo consultar una tabla con paginación.

- Implemente la paginación para los resultados de las consultas de DynamoDB.
- Utilice LastEvaluatedKey para recuperar las páginas siguientes.
- Controle el número de elementos por página con el parámetro Limit.

Java

SDK para Java 2.x

Consulte una tabla de DynamoDB con paginación mediante AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    public List<Map<String, AttributeValue>> queryWithPagination(
        final String tableName, final String partitionKeyName, final String
 partitionKeyValue, final int pageSize) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validatePositiveInteger("Page size", pageSize);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        QueryRequest.Builder queryRequestBuilder = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
```

```
            .expressionAttributeValues(expressionAttributeValues)
            .limit(pageSize);

        // List to store all items from all pages
        final List<Map<String, AttributeValue>> allItems = new ArrayList<>();

        // Map to store the last evaluated key for pagination
        Map<String, AttributeValue> lastEvaluatedKey = null;
        int pageNumber = 1;

        try {
            do {
                // If we have a last evaluated key, use it for the next page
                if (lastEvaluatedKey != null) {
                    queryRequestBuilder.exclusiveStartKey(lastEvaluatedKey);
                }

                // Execute the query
                final QueryResponse response =
dynamoDbClient.query(queryRequestBuilder.build());

                // Process the current page of results
                final List<Map<String, AttributeValue>> pageItems =
response.items();
                allItems.addAll(pageItems);

                // Get the last evaluated key for the next page
                lastEvaluatedKey = response.lastEvaluatedKey();
                if (lastEvaluatedKey != null && lastEvaluatedKey.isEmpty()) {
                    lastEvaluatedKey = null;
                }

                System.out.println("Page " + pageNumber + ": Retrieved " +
pageItems.size() + " items (Running total: "
                    + allItems.size() + ")");

                pageNumber++;

            } while (lastEvaluatedKey != null);

            System.out.println("Query with pagination complete. Retrieved a total
of " + allItems.size()
                + " items across " + (pageNumber - 1) + " pages");
```

```
            return allItems;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with pagination: " +
e.getMessage());
            throw e;
        }
    }
```

Muestra cómo consultar una tabla de DynamoDB con paginación.

```
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <tableName> <partitionKeyName> <partitionKeyValue> [pageSize]
[region]
                Where:
                    tableName - The Amazon DynamoDB table to query.
                    partitionKeyName - The name of the partition key attribute.
                    partitionKeyValue - The value of the partition key to query.
                    pageSize (optional) - The maximum number of items to return
per page. (Default: 10)
                    region (optional) - The AWS region where the table exists.
(Default: us-east-1)
                """;

        if (args.length < 3) {
            System.out.println(usage);
            System.exit(1);
        }

        final String tableName = args[0];
        final String partitionKeyName = args[1];
        final String partitionKeyValue = args[2];
        final int pageSize = args.length > 3 ? Integer.parseInt(args[3]) : 10;
        final Region region = args.length > 4 ? Region.of(args[4]) :
Region.US_EAST_1;
```

```
        System.out.println("Querying items with pagination (page size: " +
pageSize + ")");

        try {
            // Using the builder pattern to create and execute the query
            final List<Map<String, AttributeValue>> allItems = new
PaginationQueryBuilder()
                .withTableName(tableName)
                .withPartitionKeyName(partitionKeyName)
                .withPartitionKeyValue(partitionKeyValue)
                .withPageSize(pageSize)
                .withRegion(region)
                .executeWithPagination();

            // Process the results
            System.out.println("\nSummary: Retrieved a total of " +
allItems.size() + " items");

            // Display the first few items as a sample
            final int sampleSize = Math.min(5, allItems.size());
            if (sampleSize > 0) {
                System.out.println("\nSample of retrieved items (first " +
sampleSize + "):");
                for (int i = 0; i < sampleSize; i++) {
                    System.out.println(allItems.get(i));
                }

                if (allItems.size() > sampleSize) {
                    System.out.println("... and " + (allItems.size() -
sampleSize) + " more items");
                }
            }
        } catch (IllegalArgumentException e) {
            System.err.println("Invalid input: " + e.getMessage());
            System.exit(1);
        } catch (ResourceNotFoundException e) {
            System.err.println("Table not found: " + tableName);
            System.exit(1);
        } catch (DynamoDbException e) {
            System.err.println("DynamoDB error: " + e.getMessage());
            System.exit(1);
        } catch (Exception e) {
            System.err.println("Unexpected error: " + e.getMessage());
            System.exit(1);
```

```
        }
    }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte una tabla de DynamoDB con paginación mediante AWS SDK para JavaScript.

```
/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  pageSize = 25
) {
  try {
```

```
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize variables for pagination
    let lastEvaluatedKey = undefined;
    const allItems = [];
    let pageCount = 0;

    // Loop until all pages are retrieved
    do {
      // Construct the query input
      const input = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pkValue",
        Limit: pageSize,
        ExpressionAttributeNames: {
          "#pk": partitionKeyName
        },
        ExpressionAttributeValues: {
          ":pkValue": { S: partitionKeyValue }
        }
      };

      // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous
query
      if (lastEvaluatedKey) {
        input.ExclusiveStartKey = lastEvaluatedKey;
      }

      // Execute the query
      const command = new QueryCommand(input);
      const response = await client.send(command);

      // Process the current page of results
      pageCount++;
      console.log(`Processing page ${pageCount} with ${response.Items.length}
items`);

      // Add the items from this page to our collection
      if (response.Items && response.Items.length > 0) {
        allItems.push(...response.Items);
      }

      // Get the LastEvaluatedKey for the next page
```

```
      lastEvaluatedKey = response.LastEvaluatedKey;

    } while (lastEvaluatedKey); // Continue until there are no more pages

    console.log(`Query complete. Retrieved ${allItems.length} items in
 ${pageCount} pages.`);
    return allItems;
  } catch (error) {
    console.error(`Error querying with pagination: ${error}`);
    throw error;
  }
}

/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to
 retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */

module.exports = { queryWithPagination };
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte una tabla de DynamoDB con paginación mediante AWS SDK for Python (Boto3).

```
import boto3
from boto3.dynamodb.conditions import Key


def query_with_pagination(
    table_name, partition_key_name, partition_key_value, page_size=25,
 max_pages=None
):
    """
    Query a DynamoDB table with pagination to handle large result sets.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        page_size (int, optional): The number of items to return per page.
 Defaults to 25.
        max_pages (int, optional): The maximum number of pages to retrieve. If
 None, retrieves all pages.

    Returns:
        list: All items retrieved from the query across all pages.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
    page_count = 0
    all_items = []

    # Paginate through the results
    while True:
        # Check if we've reached the maximum number of pages
        if max_pages is not None and page_count >= max_pages:
            break
```

```
        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
 Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
 query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Process the current page of results
        items = response.get("Items", [])
        all_items.extend(items)

        # Update pagination tracking
        page_count += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if not last_evaluated_key:
            break

    return all_items


def query_with_pagination_generator(
    table_name, partition_key_name, partition_key_value, page_size=25
):
    """
    Query a DynamoDB table with pagination using a generator to handle large
 result sets.
    This approach is memory-efficient as it yields one page at a time.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
```

```
        page_size (int, optional): The number of items to return per page.
Defaults to 25.

    Yields:
        tuple: A tuple containing (items, page_number, last_page) where:
            - items is a list of items for the current page
            - page_number is the current page number (starting from 1)
            - last_page is a boolean indicating if this is the last page
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Initialize variables for pagination
    last_evaluated_key = None
    page_number = 0

    # Paginate through the results
    while True:
        # Prepare the query parameters
        query_params = {
            "KeyConditionExpression":
Key(partition_key_name).eq(partition_key_value),
            "Limit": page_size,
        }

        # Add the ExclusiveStartKey if we have a LastEvaluatedKey from a previous
query
        if last_evaluated_key:
            query_params["ExclusiveStartKey"] = last_evaluated_key

        # Execute the query
        response = table.query(**query_params)

        # Get the current page of results
        items = response.get("Items", [])
        page_number += 1

        # Get the LastEvaluatedKey for the next page, if any
        last_evaluated_key = response.get("LastEvaluatedKey")

        # Determine if this is the last page
        is_last_page = last_evaluated_key is None
```

```
        # Yield the current page of results
        yield (items, page_number, is_last_page)

        # If there's no LastEvaluatedKey, we've reached the end of the results
        if is_last_page:
            break
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Consulta de una tabla de DynamoDB con lecturas altamente coherentes mediante un SDK de AWS

En los siguientes ejemplos de código se muestra cómo consultar una tabla con lecturas altamente coherentes.

- Configure el nivel de coherencia de las consultas de DynamoDB.
- Utilice lecturas altamente coherentes para obtener los datos más actualizados.
- Comprenda las ventajas y desventajas entre la coherencia eventual y la coherencia sólida.

Java

SDK para Java 2.x

Consulte una tabla de DynamoDB con coherencia de lectura configurable mediante AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
```

```
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithConsistentReads(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final boolean useConsistentRead) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .consistentRead(useConsistentRead)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query successful. Found {0} items",
 response.count());
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
```

```
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying with consistent reads", e);
            throw e;
        }
    }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte una tabla de DynamoDB con coherencia de lectura configurable mediante AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
```

```
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
      },
      ConsistentRead: useConsistentRead
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
  }
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte una tabla de DynamoDB con la opción de lecturas altamente coherentes mediante AWS SDK for Python (Boto3).

```
import time

import boto3
from boto3.dynamodb.conditions import Key


def query_with_consistent_read(
    table_name,
    partition_key_name,
    partition_key_value,
```

```
    sort_key_name=None,
    sort_key_value=None,
    consistent_read=True,
):
    """
    Query a DynamoDB table with the option for strongly consistent reads.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str, optional): The name of the sort key attribute.
        sort_key_value (str, optional): The value of the sort key to query.
        consistent_read (bool, optional): Whether to use strongly consistent
 reads. Defaults to True.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the key condition expression
    key_condition = Key(partition_key_name).eq(partition_key_value)

    if sort_key_name and sort_key_value:
        key_condition = key_condition & Key(sort_key_name).eq(sort_key_value)

    # Perform the query with the consistent read option
    response = table.query(KeyConditionExpression=key_condition,
 ConsistentRead=consistent_read)

    return response
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Consulta de datos de DynamoDB mediante instrucciones SELECT de PartiQL con un AWS SDK

En el siguiente ejemplo de código se muestra cómo consultar datos mediante instrucciones SELECT de PartiQL.

JavaScript

SDK para JavaScript (v3)

Consulte elementos de una tabla de DynamoDB mediante instrucciones SELECT de PartiQL con AWS SDK para JavaScript.

```
/**
 * This example demonstrates how to query items from a DynamoDB table using
 PartiQL.
 * It shows different ways to select data with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Select all items from a DynamoDB table using PartiQL.
 * Note: This should be used with caution on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @returns The response from the ExecuteStatementCommand
 */
export const selectAllItems = async (tableName: string) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
```

```
    Statement: `SELECT * FROM "${tableName}"`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Select an item by its primary key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemByPartitionKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
    Parameters: [partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving item:", err);
    throw err;
  }
};
```

```
/**
 * Select an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemByCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
 ${sortKeyName} = ?`,
    Parameters: [partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving item:", err);
    throw err;
  }
};

/**
 * Select items using a filter condition with PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
```

```
export const selectItemsWithFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Select items using a begins_with function for prefix matching.
 * This is useful for querying hierarchical data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param attributeName - The attribute to check for prefix
 * @param prefix - The prefix to match
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsByPrefix = async (
  tableName: string,
  attributeName: string,
  prefix: string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE
 begins_with(${attributeName}, ?)`,
    Parameters: [prefix],
```

```
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Select items using a between condition for range queries.
 *
 * @param tableName - The name of the DynamoDB table
 * @param attributeName - The attribute to check for range
 * @param startValue - The start value of the range
 * @param endValue - The end value of the range
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsByRange = async (
  tableName: string,
  attributeName: string,
  startValue: number | string,
  endValue: number | string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${attributeName} BETWEEN ?
 AND ?`,
    Parameters: [startValue, endValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
```

```
};

/**
 * Example usage showing how to select items with different index types
 */
export const selectExamples = async () => {
  // Select all items from a table (use with caution on large tables)
  await selectAllItems("UsersTable");

  // Select by partition key (simple primary key)
  await selectItemByPartitionKey("UsersTable", "userId", "user123");

  // Select by composite key (partition key + sort key)
  await selectItemByCompositeKey("OrdersTable", "orderId", "order456",
 "productId", "prod789");

  // Select with a filter condition (can use any attribute)
  await selectItemsWithFilter("UsersTable", "userType", "premium");

  // Select items with a prefix (useful for hierarchical data)
  await selectItemsByPrefix("ProductsTable", "category", "electronics");

  // Select items within a range (useful for numeric or date ranges)
  await selectItemsByRange("OrdersTable", "orderDate", "2023-01-01",
 "2023-12-31");
};
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para JavaScript.
  - BatchExecuteStatement
  - ExecuteStatement

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Consulta de elementos de TTL en una tabla de DynamoDB con un AWS SDK

En los siguientes ejemplos de código, se muestra cómo consultar elementos de TTL.

Java

SDK para Java 2.x

Consulte una expresión filtrada para recopilar elementos de TTL en una tabla de DynamoDB mediante AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.util.Map;
import java.util.Optional;

        final QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .filterExpression(FILTER_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try (DynamoDbClient ddb = dynamoDbClient != null
            ? dynamoDbClient
            : DynamoDbClient.builder().region(region).build()) {
            final QueryResponse response = ddb.query(request);
            System.out.println("Query successful. Found " + response.count() + "
 items that have not expired yet.");

            // Print each item
            response.items().forEach(item -> {
                System.out.println("Item: " + item);
            });

            return 0;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
```

```
            throw e;
        }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte una expresión filtrada para recopilar elementos de TTL en una tabla de DynamoDB mediante AWS SDK para JavaScript.

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const queryFiltered = async (tableName, primaryKey, region = 'us-east-1')
 => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);

    const params = {
        TableName: tableName,
        KeyConditionExpression: "#pk = :pk",
        FilterExpression: "#ea > :ea",
        ExpressionAttributeNames: {
            "#pk": "primaryKey",
            "#ea": "expireAt"
        },
        ExpressionAttributeValues: marshall({
            ":pk": primaryKey,
            ":ea": currentTime
        })
    };

    try {
        const { Items } = await client.send(new QueryCommand(params));
```

```
        Items.forEach(item => {
            console.log(unmarshall(item))
        });
        return Items;
    } catch (err) {
        console.error(`Error querying items: ${err}`);
        throw err;
    }
}

// Example usage (commented out for testing)
// queryFiltered('your-table-name', 'your-partition-key-value');
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte una expresión filtrada para recopilar elementos de TTL en una tabla de DynamoDB mediante AWS SDK for Python (Boto3).

```
from datetime import datetime

import boto3


def query_dynamodb_items(table_name, partition_key):
    """

    :param table_name: Name of the DynamoDB table
    :param partition_key:
    :return:
    """
    try:
        # Initialize a DynamoDB resource
        dynamodb = boto3.resource("dynamodb", region_name="us-east-1")

        # Specify your table
        table = dynamodb.Table(table_name)
```

```
        # Get the current time in epoch format
        current_time = int(datetime.now().timestamp())

        # Perform the query operation with a filter expression to exclude expired
 items
        # response = table.query(
        #
 KeyConditionExpression=boto3.dynamodb.conditions.Key('partitionKey').eq(partition_key),
        #
 FilterExpression=boto3.dynamodb.conditions.Attr('expireAt').gt(current_time)
        # )
        response = table.query(

 KeyConditionExpression=dynamodb.conditions.Key("partitionKey").eq(partition_key),

 FilterExpression=dynamodb.conditions.Attr("expireAt").gt(current_time),
        )

        # Print the items that are not expired
        for item in response["Items"]:
            print(item)

    except Exception as e:
        print(f"Error querying items: {e}")


# Call the function with your values
query_dynamodb_items("Music", "your-partition-key-value")
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Consulta de tablas de DynamoDB mediante patrones de fecha y hora con un SDK de AWS

En los siguientes ejemplos de código se muestra cómo consultar tablas con patrones de fecha y hora.

- Almacene y consulte valores de fecha y hora en DynamoDB.
- Implemente consultas de intervalos de fechas mediante claves de clasificación.
- Formatee las cadenas de fecha para que las consultas sean eficaces.

Java

SDK para Java 2.x

Consulte mediante intervalos de fechas en las claves de clasificación con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.LocalDate;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

    public QueryResponse queryWithDateRange(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final LocalDate startDate,
        final LocalDate endDate) {

        // Focus on query logic, assuming parameters are valid
        if (startDate == null || endDate == null) {
```

```
            throw new IllegalArgumentException("Start date and end date cannot be
null");
        }

        if (endDate.isBefore(startDate)) {
            throw new IllegalArgumentException("End date must be after start
date");
        }

        // Format dates as ISO strings for DynamoDB (using just the date part)
        final String formattedStartDate = startDate.toString();
        final String formattedEndDate = endDate.toString();

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_SK, dateKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_START_DATE,
            AttributeValue.builder().s(formattedStartDate).build());
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_END_DATE,
            AttributeValue.builder().s(formattedEndDate).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            LOGGER.log(Level.INFO, "Query by date range successful. Found {0}
items", response.count());
```

```
            return response;
        } catch (ResourceNotFoundException e) {
            LOGGER.log(Level.SEVERE, "Table not found: {0}", tableName);
            throw e;
        } catch (DynamoDbException e) {
            LOGGER.log(Level.SEVERE, "Error querying by date range: {0}",
 e.getMessage());
            throw e;
        }
    }
```

Consulte mediante variables de fecha y hora con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryWithDateTime(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final String startDate,
        final String endDate) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateDateRangeParameters(dateKeyName, startDate,
 endDate);
        CodeSampleUtils.validateDateFormat("Start date", startDate);
        CodeSampleUtils.validateDateFormat("End date", endDate);

        // Create expression attribute names for the column names
```

```
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
partitionKeyName);
        expressionAttributeNames.put("#dateKey", dateKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            ":startDate", AttributeValue.builder().s(startDate).build());
        expressionAttributeValues.put(
            ":endDate", AttributeValue.builder().s(endDate).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query successful. Found " + response.count() + "
items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with date range: " +
e.getMessage());
            throw e;
        }
    }
```

Consulte dentro de intervalos de fechas en marcas temporales en formato de hora Unix con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.HashMap;
import java.util.Map;

    public QueryResponse queryWithDateTimeEpoch(
        final String tableName,
        final String partitionKeyName,
        final String partitionKeyValue,
        final String dateKeyName,
        final long startEpoch,
        final long endEpoch) {

        CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
 partitionKeyValue);
        CodeSampleUtils.validateStringParameter("Date key name", dateKeyName);
        CodeSampleUtils.validateEpochTimestamp("Start epoch", startEpoch);
        CodeSampleUtils.validateEpochTimestamp("End epoch", endEpoch);

        // Create expression attribute names for the column names
        final Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put(EXPRESSION_ATTRIBUTE_NAME_PK,
 partitionKeyName);
        expressionAttributeNames.put("#dateKey", dateKeyName);

        // Create expression attribute values for the column values
        final Map<String, AttributeValue> expressionAttributeValues = new
 HashMap<>();
        expressionAttributeValues.put(
            EXPRESSION_ATTRIBUTE_VALUE_PK,
            AttributeValue.builder().s(partitionKeyValue).build());
        expressionAttributeValues.put(
            ":startDate",
 AttributeValue.builder().n(String.valueOf(startEpoch)).build());
        expressionAttributeValues.put(
```

```
            ":endDate",
 AttributeValue.builder().n(String.valueOf(endEpoch)).build());

        // Create the query request
        final QueryRequest queryRequest = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression(KEY_CONDITION_EXPRESSION)
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final QueryResponse response = dynamoDbClient.query(queryRequest);
            System.out.println("Query successful. Found " + response.count() + "
 items");
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format("Error: The Amazon DynamoDB table \"%s\" can't be
 found.\n", tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println("Error querying with epoch timestamps: " +
 e.getMessage());
            throw e;
        }
    }
```

Consulte dentro de rangos de fechas mediante objetos de LocalDateTime con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.HashMap;
import java.util.Map;
```

```
   public QueryResponse queryWithDateTimeLocalDateTime(
       final String tableName,
       final String partitionKeyName,
       final String partitionKeyValue,
       final String dateKeyName,
       final LocalDateTime startDateTime,
       final LocalDateTime endDateTime) {

       CodeSampleUtils.validateTableParameters(tableName, partitionKeyName,
partitionKeyValue);
       CodeSampleUtils.validateStringParameter("Date key name", dateKeyName);
       if (startDateTime == null || endDateTime == null) {
           throw new IllegalArgumentException("Start and end LocalDateTime must
not be null");
       }

       // Convert LocalDateTime to ISO-8601 strings in UTC with the correct
format
       final String startDate =
startDateTime.atZone(ZoneOffset.UTC).format(DATE_TIME_FORMATTER);
       final String endDate =
endDateTime.atZone(ZoneOffset.UTC).format(DATE_TIME_FORMATTER);

       return queryWithDateTime(tableName, partitionKeyName, partitionKeyValue,
dateKeyName, startDate, endDate);
   }
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Consulte mediante intervalos de fechas en las claves de clasificación con AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
```

```
 * Queries a DynamoDB table for items within a specific date range on the sort
 key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
 attribute)
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRangeOnSortKey(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
 AND :endDate',
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
```

```
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range on sort key: ${error}`);
    throw error;
  }
}
```

Consulte mediante variables de fecha y hora con AWS SDK para JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} dateKeyName - The name of the date attribute to filter on
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRange(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  dateKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
```

```
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: `#pk = :pkValue AND #dateAttr BETWEEN :startDate
 AND :endDate`,
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#dateAttr": dateKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":startDate": { S: formattedStartDate },
        ":endDate": { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range: ${error}`);
    throw error;
  }
}
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Consulte mediante intervalos de fechas en las claves de clasificación con AWS SDK for Python (Boto3).

```
from datetime import datetime, timedelta

import boto3
from boto3.dynamodb.conditions import Key
```

```
def query_with_date_range(
    table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
):
    """
    Query a DynamoDB table with a date range on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        start_date (datetime): The start date for the query range.
        end_date (datetime): The end date for the query range.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Format the date values as ISO 8601 strings
    # DynamoDB works well with ISO format for date values
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key using BETWEEN operator
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(
        KeyConditionExpression=key_condition,
        ExpressionAttributeValues={
            ":pk_val": partition_key_value,
            ":start_date": start_date_str,
            ":end_date": end_date_str,
        },
    )
```

```
    return response


def query_with_date_range_by_month(
    table_name, partition_key_name, partition_key_value, sort_key_name, year,
 month
):
    """
    Query a DynamoDB table for a specific month's data.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date
 values).
        year (int): The year to query.
        month (int): The month to query (1-12).

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Calculate the start and end dates for the specified month
    if month == 12:
        next_year = year + 1
        next_month = 1
    else:
        next_year = year
        next_month = month + 1

    start_date = datetime(year, month, 1)
    end_date = datetime(next_year, next_month, 1) - timedelta(microseconds=1)

    # Format the date values as ISO 8601 strings
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key
```

```
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(KeyConditionExpression=key_condition)

    return response
```

Consulte mediante variables de fecha y hora con AWS SDK for Python (Boto3).

```
from datetime import datetime, timedelta

import boto3
from boto3.dynamodb.conditions import Key


def query_with_datetime(
    table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
):
    """
    Query a DynamoDB table with a date range filter on the sort key.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        sort_key_name (str): The name of the sort key attribute (containing date/
time values).
        start_date (datetime): The start date/time for the query range.
        end_date (datetime): The end date/time for the query range.

    Returns:
        dict: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)
```

```
    # Format the date/time values as ISO 8601 strings
    # DynamoDB works well with ISO format for date/time values
    start_date_str = start_date.isoformat()
    end_date_str = end_date.isoformat()

    # Perform the query with a date range on the sort key
    key_condition = Key(partition_key_name).eq(partition_key_value) &
 Key(sort_key_name).between(
        start_date_str, end_date_str
    )

    response = table.query(
        KeyConditionExpression=key_condition,
        ExpressionAttributeValues={
            ":pk_val": partition_key_value,
            ":start_date": start_date_str,
            ":end_date": end_date_str,
        },
    )

    return response



def example_usage():
    """Example of how to use the query_with_datetime function."""
    # Example parameters
    table_name = "Events"
    partition_key_name = "EventType"
    partition_key_value = "UserLogin"
    sort_key_name = "Timestamp"

    # Create date/time variables for the query
    end_date = datetime.now()
    start_date = end_date - timedelta(days=7)  # Query events from the last 7
 days

    print(f"Querying events from {start_date.isoformat()} to
 {end_date.isoformat()}")

    # Execute the query
    response = query_with_datetime(
        table_name, partition_key_name, partition_key_value, sort_key_name,
 start_date, end_date
```

```
    )

    # Process the results
    items = response.get("Items", [])
    print(f"Found {len(items)} items")

    for item in items:
        print(f"Event: {item}")
```

- Para obtener información sobre la API, consulte Query en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Guarde EXIF y otra información de la imagen con un SDK de AWS

En el siguiente ejemplo de código, se muestra cómo:

- Obtenga información EXIF de un archivo JPG, JPEG o PNG.
- Subir el archivo de imagen en un bucket de Amazon S3.
- Usar Amazon Rekognition para identificar los tres atributos principales (etiquetas) en el archivo.
- Agregar la información EXIF y de etiquetas a una tabla de Amazon DynamoDB de la región.

Rust

SDK para Rust

Obtenga información EXIF de un archivo JPG, JPEG o PNG, cargue el archivo de imagen en un bucket de Amazon S3, utilice Amazon Rekognition para identificar los tres atributos principales (etiquetas de Amazon Rekognition) en el archivo y añada la información EXIF y de etiquetas a una tabla de Amazon DynamoDB de la región.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- DynamoDB
- Amazon Rekognition
- Amazon S3

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Configuración del control de acceso basado en atributos para DynamoDB mediante AWS Command Line Interface v2

El siguiente ejemplo de código muestra cómo implementar el control de acceso basado en atributos (ABAC) para DynamoDB.

- Cree una política de IAM para ABAC.
- Cree tablas con etiquetas para diferentes departamentos.
- Enumere y filtre las tablas en función de las etiquetas.

Bash

AWS CLI con Bash script

Cree una política de IAM para ABAC.

```
# Step 1: Create a policy document for ABAC
cat > abac-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:Scan"
      ],
```

```
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Department": "${aws:PrincipalTag/Department}"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Department": "${aws:PrincipalTag/Department}",
          "aws:ResourceTag/Environment": "Development"
        }
      }
    }
  ]
}
EOF

# Step 2: Create the IAM policy
aws iam create-policy \
    --policy-name DynamoDBDepartmentBasedAccess \
    --policy-document file://abac-policy.json
```

Cree tablas con etiquetas para diferentes departamentos.

```
# Create a DynamoDB table with tags for ABAC
aws dynamodb create-table \
    --table-name FinanceData \
    --attribute-definitions \
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
```

```
    --tags \
        Key=Department,Value=Finance \
        Key=Environment,Value=Development

# Create another table with different tags
aws dynamodb create-table \
    --table-name MarketingData \
    --attribute-definitions \
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
        Key=Department,Value=Marketing \
        Key=Environment,Value=Production
```

Enumere y filtre las tablas en función de las etiquetas.

```
# List all DynamoDB tables
echo "Listing all tables:"
aws dynamodb list-tables

# Get ARNs for all tables
echo -e "\nGetting ARNs for all tables:"
TABLE_ARNS=$(aws dynamodb list-tables --query "TableNames[*]" --output text |
 xargs -I {} aws dynamodb describe-table --table-name {} --query "Table.TableArn"
 --output text)

# For each table ARN, list its tags
echo -e "\nListing tags for each table:"
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    echo -e "\nTags for table: $TABLE_NAME"
    aws dynamodb list-tags-of-resource --resource-arn $ARN
done

# Example: Find tables with a specific tag
echo -e "\nFinding tables with Environment=Production tag:"
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    TAGS=$(aws dynamodb list-tags-of-resource --resource-arn $ARN --query "Tags[?
Key=='Environment' && Value=='Production']" --output text)
```

```
    if [ ! -z "$TAGS" ]; then
        echo "Table with Production tag: $TABLE_NAME"
    fi
done
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de comandos de AWS CLI.
  - CreatePolicy
  - CreateTable
  - ListTables

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Descripción del orden de las expresiones de actualización en DynamoDB con un AWS SDK

En los siguientes ejemplos de código se muestra cómo comprender el orden de las expresiones de actualización.

- Obtenga información sobre cómo DynamoDB procesa las expresiones de actualización.
- Conozca el orden de las operaciones en las expresiones de actualización.
- Conozca la evaluación de expresiones para evitar resultados inesperados.

Java

SDK para Java 2.x

Demuestre el orden de las expresiones de actualización con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
```

```
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;

    /**
     * Demonstrates the effect of update expression order.
     *
     * <p>This method shows how the order of operations in an update expression
     * affects the result of the update.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @return Map containing the results of different update orders
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Map<String, Object> demonstrateUpdateOrder(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
 AttributeValue> key) {

        Map<String, Object> results = new HashMap<>();

        try {
            // Initialize the item with a counter
            UpdateItemRequest initRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = :zero, OldCounter = :zero")
                .expressionAttributeValues(
                    Map.of(":zero", AttributeValue.builder().n("0").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            dynamoDbClient.updateItem(initRequest);

            // Example 1: SET first, then ADD
            UpdateItemRequest setFirstRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = :value ADD
 OldCounter :increment")
                .expressionAttributeValues(Map.of(
```

```
                    ":value", AttributeValue.builder().n("10").build(),
                    ":increment", AttributeValue.builder().n("5").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse setFirstResponse =
dynamoDbClient.updateItem(setFirstRequest);
            results.put("setFirstResponse", setFirstResponse);

            // Reset the item
            dynamoDbClient.updateItem(initRequest);

            // Example 2: ADD first, then SET
            UpdateItemRequest addFirstRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("ADD Counter :increment SET OldCounter
= :value")
                .expressionAttributeValues(Map.of(
                    ":value", AttributeValue.builder().n("10").build(),
                    ":increment", AttributeValue.builder().n("5").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse addFirstResponse =
dynamoDbClient.updateItem(addFirstRequest);
            results.put("addFirstResponse", addFirstResponse);

            // Reset the item
            dynamoDbClient.updateItem(initRequest);

            // Example 3: SET with multiple attributes
            UpdateItemRequest multiSetRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = :value, OldCounter = Counter")
                .expressionAttributeValues(
                    Map.of(":value", AttributeValue.builder().n("10").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse multiSetResponse =
dynamoDbClient.updateItem(multiSetRequest);
            results.put("multiSetResponse", multiSetResponse);
```

```
            // Reset the item
            dynamoDbClient.updateItem(initRequest);

            // Example 4: SET with expression using the same attribute
            UpdateItemRequest selfReferenceRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET Counter = Counter + :increment, OldCounter
= Counter")
                .expressionAttributeValues(
                    Map.of(":increment",
AttributeValue.builder().n("5").build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse selfReferenceResponse =
dynamoDbClient.updateItem(selfReferenceRequest);
            results.put("selfReferenceResponse", selfReferenceResponse);

            results.put("success", true);

        } catch (DynamoDbException e) {
            results.put("success", false);
            results.put("error", e.getMessage());
        }

        return results;
    }

    /**
     * Updates an item with SET first, then REMOVE.
     *
     * <p>This method demonstrates updating an item with SET operation first,
     * followed by a REMOVE operation.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param attributeToSet The attribute to set
     * @param setValue The value to set
     * @param attributeToRemove The attribute to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
```

```
     */
    public static UpdateItemResponse updateWithSetFirst(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String attributeToSet,
        AttributeValue setValue,
        String attributeToRemove) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #setAttr = :setValue REMOVE #removeAttr")
            .expressionAttributeNames(Map.of(
                "#setAttr", attributeToSet,
                "#removeAttr", attributeToRemove))
            .expressionAttributeValues(Map.of(":setValue", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        try {
            return dynamoDbClient.updateItem(request);
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to update item with SET first: " +
e.getMessage())
                .cause(e)
                .build();
        }
    }

    /**
     * Updates an item with REMOVE first, then SET.
     *
     * <p>This method demonstrates updating an item with REMOVE operation first,
     * followed by a SET operation.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param attributeToSet The attribute to set
     * @param setValue The value to set
```

```
     * @param attributeToRemove The attribute to remove
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateWithRemoveFirst(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String attributeToSet,
        AttributeValue setValue,
        String attributeToRemove) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("REMOVE #removeAttr SET #setAttr = :setValue")
            .expressionAttributeNames(Map.of(
                "#setAttr", attributeToSet,
                "#removeAttr", attributeToRemove))
            .expressionAttributeValues(Map.of(":setValue", setValue))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        try {
            return dynamoDbClient.updateItem(request);
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to update item with REMOVE first: " +
e.getMessage())
                .cause(e)
                .build();
        }
    }

    /**
     * Updates an item with all operation types in a specific order.
     *
     * <p>This method demonstrates using all operation types (SET, REMOVE, ADD,
DELETE)
     * in a specific order in a single update expression.
     *
     * @param dynamoDbClient The DynamoDB client
```

```
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateWithAllOperationTypes(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #stringAttr = :stringVal, #mapAttr.#nestedAttr
= :nestedVal " + "REMOVE #oldAttr "
                + "ADD #counterAttr :increment "
                + "DELETE #stringSetAttr :stringSetVal")
            .expressionAttributeNames(Map.of(
                "#stringAttr", "StringAttribute",
                "#mapAttr", "MapAttribute",
                "#nestedAttr", "NestedAttribute",
                "#oldAttr", "OldAttribute",
                "#counterAttr", "CounterAttribute",
                "#stringSetAttr", "StringSetAttribute"))
            .expressionAttributeValues(Map.of(
                ":stringVal", AttributeValue.builder().s("New Value").build(),
                ":nestedVal", AttributeValue.builder().s("Nested Value").build(),
                ":increment", AttributeValue.builder().n("1").build(),
                ":stringSetVal", AttributeValue.builder().ss("Value1").build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        try {
            return dynamoDbClient.updateItem(request);
        } catch (DynamoDbException e) {
            throw DynamoDbException.builder()
                .message("Failed to update item with all operation types: " +
e.getMessage())
                .cause(e)
                .build();
        }
    }
```

```
   /**
    * Gets the current state of an item.
    *
    * <p>Helper method to retrieve the current state of an item.
    *
    * @param dynamoDbClient The DynamoDB client
    * @param tableName The name of the DynamoDB table
    * @param key The key of the item to get
    * @return The item or null if not found
    * @throws DynamoDbException if an error occurs during the operation
    */
   public static Map<String, AttributeValue> getItem(
       DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {

       // Define the get parameters
       GetItemRequest request =
           GetItemRequest.builder().tableName(tableName).key(key).build();

       // Perform the get operation
       try {
           GetItemResponse response = dynamoDbClient.getItem(request);

           // Return the item if it exists, otherwise null
           return response.item();
       } catch (DynamoDbException e) {
           throw DynamoDbException.builder()
               .message("Failed to get item: " + e.getMessage())
               .cause(e)
               .build();
       }
   }
```

Ejemplo de uso del orden de las expresiones de actualización con AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
       // Example key
       Map<String, AttributeValue> key = new HashMap<>();
       key.put("ProductId", AttributeValue.builder().s("P12345").build());

       System.out.println("Demonstrating update expression order in DynamoDB");
```

```
        try {
            // Example 1: Demonstrate update order effects
            System.out.println("\nExample 1: Demonstrating update order
effects");
            Map<String, Object> orderResults =
demonstrateUpdateOrder(dynamoDbClient, tableName, key);

            if ((boolean) orderResults.get("success")) {
                System.out.println("SET first, then ADD:");
                System.out.println("  " + orderResults.get("setFirstResponse"));

                System.out.println("ADD first, then SET:");
                System.out.println("  " + orderResults.get("addFirstResponse"));

                System.out.println("SET with multiple attributes:");
                System.out.println("  " + orderResults.get("multiSetResponse"));

                System.out.println("SET with self-reference:");
                System.out.println("  " +
orderResults.get("selfReferenceResponse"));
            } else {
                System.out.println("Error: " + orderResults.get("error"));
            }

            // Example 2: Update with SET first, then REMOVE
            System.out.println("\nExample 2: Update with SET first, then
REMOVE");
            UpdateItemResponse setFirstResponse = updateWithSetFirst(
                dynamoDbClient,
                tableName,
                key,
                "Status",
                AttributeValue.builder().s("Active").build(),
                "OldStatus");

            System.out.println("Updated attributes: " +
setFirstResponse.attributes());

            // Example 3: Update with REMOVE first, then SET
            System.out.println("\nExample 3: Update with REMOVE first, then
SET");
            UpdateItemResponse removeFirstResponse = updateWithRemoveFirst(
                dynamoDbClient,
```

```
                tableName,
                key,
                "Status",
                AttributeValue.builder().s("Inactive").build(),
                "OldStatus");

            System.out.println("Updated attributes: " +
removeFirstResponse.attributes());

            // Example 4: Update with all operation types
            System.out.println("\nExample 4: Update with all operation types");
            UpdateItemResponse allOpsResponse =
updateWithAllOperationTypes(dynamoDbClient, tableName, key);

            System.out.println("Updated attributes: " +
allOpsResponse.attributes());

            // Example 5: Get the current state of the item
            System.out.println("\nExample 5: Current state of the item");
            Map<String, AttributeValue> item = getItem(dynamoDbClient, tableName,
key);

            if (item != null) {
                System.out.println("Item: " + item);
            } else {
                System.out.println("Item not found");
            }

            // Explain update expression order
            System.out.println("\nKey points about update expression order in
DynamoDB:");
            System.out.println("1. Update expressions are processed in this
order: SET, REMOVE, ADD, DELETE");
            System.out.println("2. Within each clause, operations are processed
from left to right");
            System.out.println("3. SET operations use the item state before any
updates in the expression");
            System.out.println("4. When an attribute is referenced multiple
times, the first operation wins");
            System.out.println("5. To reference a new value, split the update
into multiple operations");
            System.out.println("6. The order of clauses in the expression doesn't
change the evaluation order");
```

```
            System.out.println("7. For complex updates, consider using multiple
 separate update operations");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Demuestre el orden de las expresiones de actualización con AWS SDK para JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an item with multiple actions in a single update expression.
 *
 * This function demonstrates how to use multiple actions in a single update
 expression
 * and how DynamoDB processes these actions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {string} updateExpression - The update expression with multiple actions
 * @param {Object} [expressionAttributeNames] - Expression attribute name
 placeholders
 * @param {Object} [expressionAttributeValues] - Expression attribute value
 placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
```

```
 */
async function updateWithMultipleActions(
  config,
  tableName,
  key,
  updateExpression,
  expressionAttributeNames,
  expressionAttributeValues
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Prepare the update parameters
  const updateParams = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ReturnValues: "UPDATED_NEW"
  };

  // Add expression attribute names if provided
  if (expressionAttributeNames) {
    updateParams.ExpressionAttributeNames = expressionAttributeNames;
  }

  // Add expression attribute values if provided
  if (expressionAttributeValues) {
    updateParams.ExpressionAttributeValues = expressionAttributeValues;
  }

  // Execute the update
  const response = await docClient.send(new UpdateCommand(updateParams));

  return response;
}

/**
 * Demonstrate that variables hold copies of existing values before
 modifications.
 *
 * This function creates an item with initial values, then updates it with an
 expression
```

```
 * that uses the values of attributes before they are modified in the same
 expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateValueCopying(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = { ...key, a: 1, b: 2, c: 3 };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with an expression that uses values before they are
 modified
  // This expression removes 'a', then sets 'b' to the value of 'a', and 'c' to
 the value of 'b'
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: "REMOVE a SET b = a, c = b",
    ReturnValues: "UPDATED_NEW"
  }));
```

```
  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}

/**
 * Demonstrate the order in which different action types are processed.
 *
 * This function creates an item with initial values, then updates it with an
 expression
 * that includes multiple action types (SET, REMOVE, ADD, DELETE) to show the
 order
 * in which they are processed.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateActionOrder(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = {
    ...key,
```

```
    counter: 10,
    set_attr: new Set(["A", "B", "C"]),
    to_remove: "This will be removed",
    to_modify: "Original value"
  };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with multiple action types
  // The actions will be processed in this order: REMOVE, SET, ADD, DELETE
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: "REMOVE to_remove SET to_modify = :new_value ADD
counter :increment DELETE set_attr :elements",
    ExpressionAttributeValues: {
      ":new_value": "Updated value",
      ":increment": 5,
      ":elements": new Set(["B"])
    },
    ReturnValues: "UPDATED_NEW"
  }));

  // Step 4: Get the item to verify final state
  const responseAfter = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemAfter = responseAfter.Item || {};

  // Return the results
  return {
```

```
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}

/**
 * Update multiple attributes with a single SET action.
 *
 * This function demonstrates how to update multiple attributes in a single SET
 action,
 * which is more efficient than using multiple separate update operations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {Object} attributes - The attributes to update and their new values
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleSetActions(
  config,
  tableName,
  key,
  attributes
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = "SET ";
  const expressionAttributeValues = {};

  // Add each attribute to the update expression
  Object.entries(attributes).forEach(([attrName, attrValue], index) => {
    const valuePlaceholder = `:val${index}`;

    if (index > 0) {
      updateExpression += ", ";
    }
    updateExpression += `${attrName} = ${valuePlaceholder}`;

    expressionAttributeValues[valuePlaceholder] = attrValue;
  });
```

```
  // Execute the update
  const response = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  }));

  return response;
}

/**
 * Update an attribute with a value from another attribute or a default value.
 *
 * This function demonstrates how to use if_not_exists to conditionally copy a
 value
 * from one attribute to another, or use a default value if the source doesn't
 exist.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {string} sourceAttribute - The attribute to copy the value from
 * @param {string} targetAttribute - The attribute to update
 * @param {any} defaultValue - The default value to use if the source attribute
 doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithConditionalValueCopying(
  config,
  tableName,
  key,
  sourceAttribute,
  targetAttribute,
  defaultValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Use if_not_exists to conditionally copy the value
  const response = await docClient.send(new UpdateCommand({
```

```
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${targetAttribute} =
 if_not_exists(${sourceAttribute}, :default)`,
    ExpressionAttributeValues: {
      ":default": defaultValue
    },
    ReturnValues: "UPDATED_NEW"
  }));

  return response;
}

/**
 * Demonstrate complex update expressions with multiple operations on the same
 attribute.
 *
 * This function shows how DynamoDB processes multiple operations on the same
 attribute
 * in a single update expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 demonstration
 */
async function demonstrateMultipleOperationsOnSameAttribute(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = {
    ...key,
    counter: 10,
    list_attr: [1, 2, 3],
    map_attr: {
      nested1: "value1",
      nested2: "value2"
```

```
  }
};

await docClient.send(new PutCommand({
  TableName: tableName,
  Item: initialItem
}));

// Step 2: Get the item to verify initial state
const responseBefore = await docClient.send(new GetCommand({
  TableName: tableName,
  Key: key
}));

const itemBefore = responseBefore.Item || {};

// Step 3: Update the item with multiple operations on the same attributes
const updateResponse = await docClient.send(new UpdateCommand({
  TableName: tableName,
  Key: key,
  UpdateExpression: `
    SET counter = counter + :inc1,
        counter = counter + :inc2,
        map_attr.nested1 = :new_val1,
        map_attr.nested3 = :new_val3,
        list_attr[0] = list_attr[1],
        list_attr[1] = list_attr[2]
  `,
  ExpressionAttributeValues: {
    ":inc1": 5,
    ":inc2": 3,
    ":new_val1": "updated_value1",
    ":new_val3": "new_value3"
  },
  ReturnValues: "UPDATED_NEW"
}));

// Step 4: Get the item to verify final state
const responseAfter = await docClient.send(new GetCommand({
  TableName: tableName,
  Key: key
}));

const itemAfter = responseAfter.Item || {};
```

```
  // Return the results
  return {
    initialState: itemBefore,
    updateResponse: updateResponse,
    finalState: itemAfter
  };
}
```

Ejemplo de uso del orden de las expresiones de actualización con AWS SDK para JavaScript.

```
/**
 * Example of how to use update expression order of operations in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "OrderProcessing";

  console.log("Demonstrating update expression order of operations in DynamoDB");

  try {
    // Example 1: Demonstrating value copying in update expressions
    console.log("\nExample 1: Demonstrating value copying in update
 expressions");
    const results1 = await demonstrateValueCopying(
      config,
      tableName,
      { OrderId: "order123" }
    );

    console.log("Initial state:", JSON.stringify(results1.initialState, null,
 2));
    console.log("Update response:", JSON.stringify(results1.updateResponse, null,
 2));
    console.log("Final state:", JSON.stringify(results1.finalState, null, 2));

    console.log("\nExplanation:");
    console.log("1. The initial state had a=1, b=2, c=3");
    console.log("2. The update expression 'REMOVE a SET b = a, c = b' did the
 following:");
    console.log("   - Copied the value of 'a' (which was 1) to be used for 'b'");
```

```
   console.log("   - Copied the value of 'b' (which was 2) to be used for 'c'");
   console.log("   - Removed the attribute 'a'");
   console.log("3. The final state has b=1, c=2, and 'a' is removed");
   console.log("4. This demonstrates that DynamoDB uses the values of attributes
as they were BEFORE any modifications");

   // Example 2: Demonstrating the order of different action types
   console.log("\nExample 2: Demonstrating the order of different action
types");
   const results2 = await demonstrateActionOrder(
     config,
     tableName,
     { OrderId: "order456" }
   );

   console.log("Initial state:", JSON.stringify(results2.initialState, null,
2));
   console.log("Update response:", JSON.stringify(results2.updateResponse, null,
2));
   console.log("Final state:", JSON.stringify(results2.finalState, null, 2));

   console.log("\nExplanation:");
   console.log("1. The update expression contained multiple action types:
REMOVE, SET, ADD, DELETE");
   console.log("2. DynamoDB processes these actions in this order: REMOVE, SET,
ADD, DELETE");
   console.log("3. First, 'to_remove' was removed");
   console.log("4. Then, 'to_modify' was set to a new value");
   console.log("5. Next, 'counter' was incremented by 5");
   console.log("6. Finally, 'B' was removed from the set attribute");

   // Example 3: Updating multiple attributes in a single SET action
   console.log("\nExample 3: Updating multiple attributes in a single SET
action");
   const response3 = await updateWithMultipleSetActions(
     config,
     tableName,
     { OrderId: "order789" },
     {
       Status: "Shipped",
       ShippingDate: "2025-05-28",
       TrackingNumber: "1Z999AA10123456784"
     }
   );
```

```
   console.log("Multiple attributes updated successfully:",
JSON.stringify(response3.Attributes, null, 2));

   // Example 4: Conditional value copying with if_not_exists
   console.log("\nExample 4: Conditional value copying with if_not_exists");
   const response4 = await updateWithConditionalValueCopying(
     config,
     tableName,
     { OrderId: "order101" },
     "PreferredShippingMethod",
     "ShippingMethod",
     "Standard"
   );

   console.log("Conditional value copying result:",
JSON.stringify(response4.Attributes, null, 2));

   // Example 5: Multiple operations on the same attribute
   console.log("\nExample 5: Multiple operations on the same attribute");
   const results5 = await demonstrateMultipleOperationsOnSameAttribute(
     config,
     tableName,
     { OrderId: "order202" }
   );

   console.log("Initial state:", JSON.stringify(results5.initialState, null,
2));
   console.log("Update response:", JSON.stringify(results5.updateResponse, null,
2));
   console.log("Final state:", JSON.stringify(results5.finalState, null, 2));

   console.log("\nExplanation:");
   console.log("1. The counter was incremented twice (first by 5, then by 3) for
a total of +8");
   console.log("2. The map attribute had one value updated and a new nested
attribute added");
   console.log("3. The list attribute had values shifted (value at index 1 moved
to index 0, value at index 2 moved to index 1)");
   console.log("4. All operations within the SET action are processed from left
to right");

   // Key points about update expression order of operations
   console.log("\nKey Points About Update Expression Order of Operations:");
```

```
    console.log("1. Variables in expressions hold copies of attribute values as
 they existed BEFORE any modifications");
    console.log("2. Multiple actions in an update expression are processed in
 this order: REMOVE, SET, ADD, DELETE");
    console.log("3. Within each action type, operations are processed from left
 to right");
    console.log("4. You can reference the same attribute multiple times in an
 expression");
    console.log("5. You can use if_not_exists() to conditionally set values based
 on attribute existence");
    console.log("6. Using a single update expression with multiple actions is
 more efficient than multiple separate updates");
    console.log("7. The update expression is atomic - either all actions succeed
 or none do");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Demuestre el orden de las expresiones de actualización con AWS SDK for Python (Boto3).

```
import boto3
import json
from typing import Any, Dict, Optional


def update_with_multiple_actions(
    table_name: str,
    key: Dict[str, Any],
    update_expression: str,
    expression_attribute_names: Optional[Dict[str, str]] = None,
    expression_attribute_values: Optional[Dict[str, Any]] = None,
) -> Dict[str, Any]:
    """
```

```
    Update an item with multiple actions in a single update expression.

    This function demonstrates how to use multiple actions in a single update
expression
    and how DynamoDB processes these actions.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        update_expression (str): The update expression with multiple actions.
        expression_attribute_names (Optional[Dict[str, str]]): Expression
attribute name placeholders.
        expression_attribute_values (Optional[Dict[str, Any]]): Expression
attribute value placeholders.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Prepare the update parameters
    update_params = {
        "Key": key,
        "UpdateExpression": update_expression,
        "ReturnValues": "UPDATED_NEW",
    }

    # Add expression attribute names if provided
    if expression_attribute_names:
        update_params["ExpressionAttributeNames"] = expression_attribute_names

    # Add expression attribute values if provided
    if expression_attribute_values:
        update_params["ExpressionAttributeValues"] = expression_attribute_values

    # Execute the update
    response = table.update_item(**update_params)

    return response
```

```
def demonstrate_value_copying(table_name: str, key: Dict[str, Any]) -> Dict[str,
 Any]:
    """
    Demonstrate that variables hold copies of existing values before
 modifications.

    This function creates an item with initial values, then updates it with an
 expression
    that uses the values of attributes before they are modified in the same
 expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to create and update.

    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Step 1: Create an item with initial values
    initial_item = key.copy()
    initial_item.update({"a": 1, "b": 2, "c": 3})

    table.put_item(Item=initial_item)

    # Step 2: Get the item to verify initial state
    response_before = table.get_item(Key=key)
    item_before = response_before.get("Item", {})

    # Step 3: Update the item with an expression that uses values before they are
 modified
    # This expression removes 'a', then sets 'b' to the value of 'a', and 'c' to
 the value of 'b'
    update_response = table.update_item(
        Key=key, UpdateExpression="REMOVE a SET b = a, c = b",
 ReturnValues="UPDATED_NEW"
    )

    # Step 4: Get the item to verify final state
    response_after = table.get_item(Key=key)
    item_after = response_after.get("Item", {})
```

```
    # Return the results
    return {
        "initial_state": item_before,
        "update_response": update_response,
        "final_state": item_after,
    }


def demonstrate_action_order(table_name: str, key: Dict[str, Any]) -> Dict[str,
 Any]:
    """
    Demonstrate the order in which different action types are processed.

    This function creates an item with initial values, then updates it with an
 expression
    that includes multiple action types (SET, REMOVE, ADD, DELETE) to show the
 order
    in which they are processed.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to create and update.

    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Step 1: Create an item with initial values
    initial_item = key.copy()
    initial_item.update(
        {
            "counter": 10,
            "set_attr": set(["A", "B", "C"]),
            "to_remove": "This will be removed",
            "to_modify": "Original value",
        }
    )

    table.put_item(Item=initial_item)
```

```
    # Step 2: Get the item to verify initial state
    response_before = table.get_item(Key=key)
    item_before = response_before.get("Item", {})

    # Step 3: Update the item with multiple action types
    # The actions will be processed in this order: REMOVE, SET, ADD, DELETE
    update_response = table.update_item(
        Key=key,
        UpdateExpression="REMOVE to_remove SET to_modify = :new_value ADD
 counter :increment DELETE set_attr :elements",
        ExpressionAttributeValues={
            ":new_value": "Updated value",
            ":increment": 5,
            ":elements": set(["B"]),
        },
        ReturnValues="UPDATED_NEW",
    )

    # Step 4: Get the item to verify final state
    response_after = table.get_item(Key=key)
    item_after = response_after.get("Item", {})

    # Return the results
    return {
        "initial_state": item_before,
        "update_response": update_response,
        "final_state": item_after,
    }


def update_with_multiple_set_actions(
    table_name: str, key: Dict[str, Any], attributes: Dict[str, Any]
) -> Dict[str, Any]:
    """
    Update multiple attributes with a single SET action.

    This function demonstrates how to update multiple attributes in a single SET
 action,
    which is more efficient than using multiple separate update operations.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
```

```
        attributes (Dict[str, Any]): The attributes to update and their new
 values.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Build the update expression and expression attribute values
    update_expression = "SET "
    expression_attribute_values = {}

    # Add each attribute to the update expression
    for i, (attr_name, attr_value) in enumerate(attributes.items()):
        value_placeholder = f":val{i}"

        if i > 0:
            update_expression += ", "
        update_expression += f"{attr_name} = {value_placeholder}"

        expression_attribute_values[value_placeholder] = attr_value

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeValues=expression_attribute_values,
        ReturnValues="UPDATED_NEW",
    )

    return response


def update_with_conditional_value_copying(
    table_name: str,
    key: Dict[str, Any],
    source_attribute: str,
    target_attribute: str,
    default_value: Any,
) -> Dict[str, Any]:
    """
```

```
    Update an attribute with a value from another attribute or a default value.

    This function demonstrates how to use if_not_exists to conditionally copy a
value
    from one attribute to another, or use a default value if the source doesn't
exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        source_attribute (str): The attribute to copy the value from.
        target_attribute (str): The attribute to update.
        default_value (Any): The default value to use if the source attribute
doesn't exist.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use if_not_exists to conditionally copy the value
    response = table.update_item(
        Key=key,
        UpdateExpression=f"SET {target_attribute} =
if_not_exists({source_attribute}, :default)",
        ExpressionAttributeValues={":default": default_value},
        ReturnValues="UPDATED_NEW",
    )

    return response
```

Ejemplo de uso del orden de las expresiones de actualización con AWS SDK for Python (Boto3).

```
def example_usage():
    """Example of how to use update expression order of operations in
 DynamoDB."""
```

```
    # Example parameters
    table_name = "OrderProcessing"
    key = {"OrderId": "order123"}

    print("Example 1: Demonstrating value copying in update expressions")
    try:
        results = demonstrate_value_copying(table_name=table_name, key=key)

        print(f"Initial state: {json.dumps(results['initial_state'],
default=str)}")
        print(f"Update response: {json.dumps(results['update_response'],
default=str)}")
        print(f"Final state: {json.dumps(results['final_state'], default=str)}")

        print("\nExplanation:")
        print("1. The initial state had a=1, b=2, c=3")
        print("2. The update expression 'REMOVE a SET b = a, c = b' did the
following:")
        print("   - Copied the value of 'a' (which was 1) to be used for 'b'")
        print("   - Copied the value of 'b' (which was 2) to be used for 'c'")
        print("   - Removed the attribute 'a'")
        print("3. The final state has b=1, c=2, and 'a' is removed")
        print(
            "4. This demonstrates that DynamoDB uses the values of attributes as
they were BEFORE any modifications"
        )
    except Exception as e:
        print(f"Error demonstrating value copying: {e}")

    print("\nExample 2: Demonstrating the order of different action types")
    try:
        results = demonstrate_action_order(table_name=table_name, key={"OrderId":
"order456"})

        print(f"Initial state: {json.dumps(results['initial_state'],
default=str)}")
        print(f"Update response: {json.dumps(results['update_response'],
default=str)}")
        print(f"Final state: {json.dumps(results['final_state'], default=str)}")

        print("\nExplanation:")
        print("1. The update expression contained multiple action types: REMOVE,
SET, ADD, DELETE")
```

```
        print("2. DynamoDB processes these actions in this order: REMOVE, SET,
ADD, DELETE")
        print("3. First, 'to_remove' was removed")
        print("4. Then, 'to_modify' was set to a new value")
        print("5. Next, 'counter' was incremented by 5")
        print("6. Finally, 'B' was removed from the set attribute")
    except Exception as e:
        print(f"Error demonstrating action order: {e}")

    print("\nExample 3: Updating multiple attributes in a single SET action")
    try:
        response = update_with_multiple_set_actions(
            table_name=table_name,
            key={"OrderId": "order789"},
            attributes={
                "Status": "Shipped",
                "ShippingDate": "2025-05-14",
                "TrackingNumber": "1Z999AA10123456784",
            },
        )

        print(
            f"Multiple attributes updated successfully:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error updating multiple attributes: {e}")

    print("\nExample 4: Conditional value copying with if_not_exists")
    try:
        response = update_with_conditional_value_copying(
            table_name=table_name,
            key={"OrderId": "order101"},
            source_attribute="PreferredShippingMethod",
            target_attribute="ShippingMethod",
            default_value="Standard",
        )

        print(
            f"Conditional value copying result:
{json.dumps(response.get('Attributes', {}), default=str)}"
        )
    except Exception as e:
        print(f"Error with conditional value copying: {e}")
```

```
   print("\nKey Points About Update Expression Order of Operations:")
   print(
       "1. Variables in expressions hold copies of attribute values as they
existed BEFORE any modifications"
   )
   print(
       "2. Multiple actions in an update expression are processed in this order:
REMOVE, SET, ADD, DELETE"
   )
   print("3. Within each action type, operations are processed from left to
right")
   print("4. You can reference the same attribute multiple times in an
expression")
   print("5. You can use if_not_exists() to conditionally set values based on
attribute existence")
   print(
       "6. Using a single update expression with multiple actions is more
efficient than multiple separate updates"
   )
   print("7. The update expression is atomic - either all actions succeed or
none do")
```

- Para obtener información la API, consulte UpdateItem en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Actualización de una configuración de tabla de DynamoDB con rendimiento en caliente mediante un AWS SDK

En los siguientes ejemplos de código se muestra cómo actualizar la configuración de rendimiento en caliente de una tabla.

Java

SDK para Java 2.x

Actualice la configuración de rendimiento en caliente en una tabla de DynamoDB existente con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GlobalSecondaryIndexUpdate;
import
 software.amazon.awssdk.services.dynamodb.model.UpdateGlobalSecondaryIndexAction;
import software.amazon.awssdk.services.dynamodb.model.UpdateTableRequest;
import software.amazon.awssdk.services.dynamodb.model.WarmThroughput;

    public static WarmThroughput buildWarmThroughput(final Long
 readUnitsPerSecond, final Long writeUnitsPerSecond) {
        return WarmThroughput.builder()
            .readUnitsPerSecond(readUnitsPerSecond)
            .writeUnitsPerSecond(writeUnitsPerSecond)
            .build();
    }

    /**
     * Updates a DynamoDB table with warm throughput settings for both the table
 and a global secondary index.
     *
     * @param ddb The DynamoDB client
     * @param tableName The name of the table to update
     * @param tableReadUnitsPerSecond Read units per second for the table
     * @param tableWriteUnitsPerSecond Write units per second for the table
     * @param globalSecondaryIndexName The name of the global secondary index to
 update
     * @param globalSecondaryIndexReadUnitsPerSecond Read units per second for
 the GSI
     * @param globalSecondaryIndexWriteUnitsPerSecond Write units per second for
 the GSI
     */
    public static void updateDynamoDBTable(
        final DynamoDbClient ddb,
        final String tableName,
        final Long tableReadUnitsPerSecond,
        final Long tableWriteUnitsPerSecond,
        final String globalSecondaryIndexName,
```

```
        final Long globalSecondaryIndexReadUnitsPerSecond,
        final Long globalSecondaryIndexWriteUnitsPerSecond) {

        final WarmThroughput tableWarmThroughput =
            buildWarmThroughput(tableReadUnitsPerSecond,
tableWriteUnitsPerSecond);
        final WarmThroughput gsiWarmThroughput =
            buildWarmThroughput(globalSecondaryIndexReadUnitsPerSecond,
globalSecondaryIndexWriteUnitsPerSecond);

        final GlobalSecondaryIndexUpdate globalSecondaryIndexUpdate =
GlobalSecondaryIndexUpdate.builder()
            .update(UpdateGlobalSecondaryIndexAction.builder()
                .indexName(globalSecondaryIndexName)
                .warmThroughput(gsiWarmThroughput)
                .build())
            .build();

        final UpdateTableRequest request = UpdateTableRequest.builder()
            .tableName(tableName)
            .globalSecondaryIndexUpdates(globalSecondaryIndexUpdate)
            .warmThroughput(tableWarmThroughput)
            .build();

        try {
            ddb.updateTable(request);
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }

        System.out.println(SUCCESS_MESSAGE);
    }
```

- Para obtener información sobre la API, consulte UpdateTable en la Referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Actualice la configuración de rendimiento en caliente en una tabla de DynamoDB existente con AWS SDK para JavaScript.

```
import { DynamoDBClient, UpdateTableCommand } from "@aws-sdk/client-dynamodb";

export async function updateDynamoDBTableWarmThroughput(
  tableName,
  tableReadUnits,
  tableWriteUnits,
  gsiName,
  gsiReadUnits,
  gsiWriteUnits,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });

    // Construct the update table request
    const updateTableRequest = {
      TableName: tableName,
      GlobalSecondaryIndexUpdates: [
        {
            Update: {
                IndexName: gsiName,
                WarmThroughput: {
                    ReadUnitsPerSecond: gsiReadUnits,
                    WriteUnitsPerSecond: gsiWriteUnits,
                },
            },
        },
      ],
      WarmThroughput: {
          ReadUnitsPerSecond: tableReadUnits,
          WriteUnitsPerSecond: tableWriteUnits,
      },
    };

    const command = new UpdateTableCommand(updateTableRequest);
    const response = await ddbClient.send(command);
```

```
    console.log(`Table updated successfully! Response:
 ${JSON.stringify(response)}`);
    return response;
  } catch (error) {
    console.error(`Error updating table: ${error}`);
    throw error;
  }
}

// Example usage (commented out for testing)
/*
updateDynamoDBTableWarmThroughput(
  'example-table',
  5, 5,
  'example-index',
  2, 2
);
*/
```

- Para obtener información sobre la API, consulte UpdateTable en la Referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Actualice la configuración de rendimiento en caliente en una tabla de DynamoDB existente con AWS SDK for Python (Boto3).

```
from boto3 import client
from botocore.exceptions import ClientError


def update_dynamodb_table_warm_throughput(
    table_name,
    table_read_units,
    table_write_units,
    gsi_name,
    gsi_read_units,
    gsi_write_units,
```

```
    region_name="us-east-1",
):
    """
    Updates the warm throughput of a DynamoDB table and a global secondary index.

    :param table_name: The name of the table to update.
    :param table_read_units: The new read units per second for the table's warm
 throughput.
    :param table_write_units: The new write units per second for the table's warm
 throughput.
    :param gsi_name: The name of the global secondary index to update.
    :param gsi_read_units: The new read units per second for the GSI's warm
 throughput.
    :param gsi_write_units: The new write units per second for the GSI's warm
 throughput.
    :param region_name: The AWS Region name to target. defaults to us-east-1
    :return: The response from the update_table operation
    """
    try:
        ddb = client("dynamodb", region_name=region_name)

        # Update the table's warm throughput
        table_warm_throughput = {
            "ReadUnitsPerSecond": table_read_units,
            "WriteUnitsPerSecond": table_write_units,
        }

        # Update the global secondary index's warm throughput
        gsi_warm_throughput = {
            "ReadUnitsPerSecond": gsi_read_units,
            "WriteUnitsPerSecond": gsi_write_units,
        }

        # Construct the global secondary index update
        global_secondary_index_update = [
            {"Update": {"IndexName": gsi_name, "WarmThroughput":
 gsi_warm_throughput}}
        ]

        # Construct the update table request
        update_table_request = {
            "TableName": table_name,
            "GlobalSecondaryIndexUpdates": global_secondary_index_update,
            "WarmThroughput": table_warm_throughput,
```

```
        }

        # Update the table
        response = ddb.update_table(**update_table_request)
        print("Table updated successfully!")
        return response  # Make sure to return the response
    except ClientError as e:
        print(f"Error updating table: {e}")
        raise e
```

- Para obtener detalles de la API, consulte UpdateTable en la Referencia de la API del AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Actualización de un elemento de DynamoDB con un TTL mediante un AWS SDK

En los siguientes ejemplos de código, se muestra cómo actualizar el TTL de un elemento.

Java

SDK para Java 2.x

Actualice el TTL en un elemento de DynamoDB existente en una tabla.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.ResourceNotFoundException;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

    public UpdateItemResponse updateItemWithTTL(
```

```
        final String tableName, final String primaryKeyValue, final String
sortKeyValue) {
        // Get current time in epoch second format
        final long currentTime = System.currentTimeMillis() / 1000;

        // Calculate expiration time 90 days from now in epoch second format
        final long expireDate = currentTime + (DAYS_TO_EXPIRE * SECONDS_PER_DAY);

        // Create the key map for the item to update
        final Map<String, AttributeValue> keyMap = new HashMap<>();
        keyMap.put(PRIMARY_KEY_ATTR,
AttributeValue.builder().s(primaryKeyValue).build());
        keyMap.put(SORT_KEY_ATTR,
AttributeValue.builder().s(sortKeyValue).build());

        // Create the expression attribute values
        final Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();
        expressionAttributeValues.put(
            ":c",
AttributeValue.builder().n(String.valueOf(currentTime)).build());
        expressionAttributeValues.put(
            ":e",
AttributeValue.builder().n(String.valueOf(expireDate)).build());

        final UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(keyMap)
            .updateExpression(UPDATE_EXPRESSION)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        try {
            final UpdateItemResponse response =
dynamoDbClient.updateItem(request);
            System.out.println(String.format(SUCCESS_MESSAGE, tableName));
            return response;
        } catch (ResourceNotFoundException e) {
            System.err.format(TABLE_NOT_FOUND_ERROR, tableName);
            throw e;
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            throw e;
        }
```

```
    }
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItem = async (tableName, partitionKey, sortKey, region = 'us-
east-1') => {
    const client = new DynamoDBClient({
        region: region,
        endpoint: `https://dynamodb.${region}.amazonaws.com`
    });

    const currentTime = Math.floor(Date.now() / 1000);
    const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) / 1000);

    const params = {
        TableName: tableName,
        Key: marshall({
            partitionKey: partitionKey,
            sortKey: sortKey
        }),
        UpdateExpression: "SET updatedAt = :c, expireAt = :e",
        ExpressionAttributeValues: marshall({
            ":c": currentTime,
            ":e": expireAt
        }),
    };

    try {
        const data = await client.send(new UpdateItemCommand(params));
        const responseData = unmarshall(data.Attributes);
        console.log("Item updated successfully: %s", responseData);
        return responseData;
    } catch (err) {
```

```
        console.error("Error updating item:", err);
        throw err;
    }
}

// Example usage (commented out for testing)
// updateItem('your-table-name', 'your-partition-key-value', 'your-sort-key-
value');
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

```
from datetime import datetime, timedelta

import boto3


def update_dynamodb_item(table_name, region, primary_key, sort_key):
    """
    Update an existing DynamoDB item with a TTL.
    :param table_name: Name of the DynamoDB table
    :param region: AWS Region of the table - example `us-east-1`
    :param primary_key: one attribute known as the partition key.
    :param sort_key: Also known as a range attribute.
    :return: Void (nothing)
    """
    try:
        # Create the DynamoDB resource.
        dynamodb = boto3.resource("dynamodb", region_name=region)
        table = dynamodb.Table(table_name)

        # Get the current time in epoch second format
        current_time = int(datetime.now().timestamp())

        # Calculate the expireAt time (90 days from now) in epoch second format
        expire_at = int((datetime.now() + timedelta(days=90)).timestamp())
```

```
        table.update_item(
            Key={"partitionKey": primary_key, "sortKey": sort_key},
            UpdateExpression="set updatedAt=:c, expireAt=:e",
            ExpressionAttributeValues={":c": current_time, ":e": expire_at},
        )

        print("Item updated successfully.")
    except Exception as e:
        print(f"Error updating item: {e}")


# Replace with your own values
update_dynamodb_item(
    "your-table-name", "us-west-2", "your-partition-key-value", "your-sort-key-
value"
)
```

- Para obtener información la API, consulte UpdateItem en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Actualización de datos de DynamoDB mediante instrucciones UPDATE de PartiQL con un AWS SDK

En el siguiente ejemplo de código se muestra cómo actualizar datos mediante instrucciones UPDATE de PartiQL.

JavaScript

SDK para JavaScript (v3)

Actualice elementos en una tabla de DynamoDB mediante instrucciones UPDATE de PartiQL con AWS SDK para JavaScript.

```
/**
```

```
 * This example demonstrates how to update items in a DynamoDB table using
 PartiQL.
 * It shows different ways to update documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Update a single attribute of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const updateSingleAttribute = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
```

```
  }
};

/**
 * Update multiple attributes of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeUpdates - Object containing attribute names and their new
 values
 * @returns The response from the ExecuteStatementCommand
 */
export const updateMultipleAttributes = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeUpdates: Record<string, any>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create SET clause for each attribute
  const setClause = Object.keys(attributeUpdates)
    .map((attr, index) => `${attr} = ?`)
    .join(", ");

  // Create parameters array with attribute values followed by the partition key
 value
  const parameters = [...Object.values(attributeUpdates), partitionKeyValue];

  const params = {
    Statement: `UPDATE "${tableName}" SET ${setClause} WHERE ${partitionKeyName}
 = ?`,
    Parameters: parameters,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
```

```
  }
};

/**
 * Update an item identified by a composite key (partition key + sort key) using
 PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const updateItemWithCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number,
  attributeName: string,
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ? AND ${sortKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};
```

```
/**
 * Update an item with a condition to ensure the update only happens if a
 condition is met.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @param conditionAttribute - The attribute to check in the condition
 * @param conditionValue - The value to compare against in the condition
 * @returns The response from the ExecuteStatementCommand
 */
export const updateItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any,
  conditionAttribute: string,
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
 ${partitionKeyName} = ? AND ${conditionAttribute} = ?`,
    Parameters: [attributeValue, partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated with condition successfully");
    return data;
  } catch (err) {
    console.error("Error updating item with condition:", err);
    throw err;
  }
};

/**
 * Batch update multiple items using PartiQL.
 *
```

```
 * @param tableName - The name of the DynamoDB table
 * @param updates - Array of objects containing key and update information
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchUpdateItems = async (
  tableName: string,
  updates: Array<{
    partitionKeyName: string;
    partitionKeyValue: string | number;
    attributeName: string;
    attributeValue: any;
  }>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each update
  const statements = updates.map((update) => {
    return {
      Statement: `UPDATE "${tableName}" SET ${update.attributeName} = ? WHERE
 ${update.partitionKeyName} = ?`,
      Parameters: [update.attributeValue, update.partitionKeyValue],
    };
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch updated successfully");
    return data;
  } catch (err) {
    console.error("Error batch updating items:", err);
    throw err;
  }
};

/**
 * Example usage showing how to update items with different index types
 */
export const updateExamples = async () => {
  // Update a single attribute using a simple primary key
```

```
 await updateSingleAttribute("UsersTable", "userId", "user123", "email",
"newemail@example.com");

 // Update multiple attributes at once
 await updateMultipleAttributes("UsersTable", "userId", "user123", {
   email: "newemail@example.com",
   name: "John Smith",
   lastLogin: new Date().toISOString(),
 });

 // Update an item with a composite key (partition key + sort key)
 await updateItemWithCompositeKey(
   "OrdersTable",
   "orderId",
   "order456",
   "productId",
   "prod789",
   "quantity",
   5
 );

 // Update with a condition
 await updateItemWithCondition(
   "UsersTable",
   "userId",
   "user123",
   "userStatus",
   "active",
   "userType",
   "premium"
 );

 // Batch update multiple items
 await batchUpdateItems("UsersTable", [
   {
     partitionKeyName: "userId",
     partitionKeyValue: "user123",
     attributeName: "lastLogin",
     attributeValue: new Date().toISOString(),
   },
   {
     partitionKeyName: "userId",
     partitionKeyValue: "user456",
     attributeName: "lastLogin",
```

```
      attributeValue: new Date().toISOString(),
    },
  ]);
};
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para JavaScript.
  - BatchExecuteStatement
  - ExecuteStatement

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de API Gateway para invocar una función de Lambda

Los siguientes ejemplos de código muestran cómo crear una función de AWS Lambda invocada por Amazon API Gateway.

Java

SDK para Java 2.x

Indica cómo crear una función de AWS Lambda utilizando la API de tiempo de ejecución de Java de Lambda. Este ejemplo invoca diferentes servicios de AWS para realizar un caso de uso específico. En este ejemplo se indica cómo crear una función de Lambda invocada por Amazon API Gateway que escanea una tabla de Amazon DynamoDB en busca de aniversarios laborales y utiliza Amazon Simple Notification Service (Amazon SNS) para enviar un mensaje de texto a sus empleados que les felicite en la fecha de su primer aniversario.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda

- Amazon SNS

JavaScript

SDK para JavaScript (v3)

Indica cómo crear una función de Lambda mediante el uso de la API de tiempo de ejecución de JavaScript de Lambda. Este ejemplo invoca diferentes servicios de AWS para realizar un caso de uso específico. En este ejemplo se indica cómo crear una función de Lambda invocada por Amazon API Gateway que escanea una tabla de Amazon DynamoDB en busca de aniversarios laborales y utiliza Amazon Simple Notification Service (Amazon SNS) para enviar un mensaje de texto a sus empleados que les felicite en la fecha de su primer aniversario.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Este ejemplo también está disponible en la Guía para desarrolladores de AWS SDK para JavaScript v3.

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Python

SDK para Python (Boto3)

En este ejemplo se muestra cómo crear y utilizar una API de REST de Amazon API Gateway dirigida a una función AWS Lambda. El controlador Lambda muestra cómo enrutar según los métodos HTTP; cómo obtener datos de la cadena de consulta, el encabezado y el cuerpo; y cómo devolver una respuesta JSON.

- Implemente una función de Lambda.
- Cree una API de REST mediante API Gateway.
- Cree un recurso REST que se dirija a la función de Lambda.

- Otorgue permiso para permitir que API Gateway invoque la función de Lambda.
- Utilice el paquete Requests para enviar solicitudes a la API de REST.
- Limpie todos los recursos creados durante la demostración.

Este ejemplo se puede ver mejor en GitHub. Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de Step Functions para invocar funciones de Lambda

En los siguientes ejemplos de código se muestra cómo crear una máquina de estado de AWS Step Functions que invoque funciones de AWS Lambda en secuencia.

Java

SDK para Java 2.x

Muestra cómo crear un flujo de trabajo sin servidor de AWS con AWS Step Functions y AWS SDK for Java 2.x. Cada paso del flujo de trabajo se implementa con una función de AWS Lambda.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Utilizar un modelo de documento para DynamoDB mediante un AWS SDK

En el siguiente ejemplo de código se muestra cómo realizar operaciones de creación, lectura, actualización y eliminación (CRUD) y por lotes mediante un modelo de documento para DynamoDB y un AWS SDK.

Para obtener más información, consulte Document model (Modelo de documento).

.NET

SDK para .NET

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Realice operaciones de CRUD con un modelo de documento.

```
/// <summary>
/// Performs CRUD operations on an Amazon DynamoDB table.
/// </summary>
public class MidlevelItemCRUD
{
    public static async Task Main()
    {
        var tableName = "ProductCatalog";
        var sampleBookId = 555;

        var client = new AmazonDynamoDBClient();
        var productCatalog = LoadTable(client, tableName);

        await CreateBookItem(productCatalog, sampleBookId);
        RetrieveBook(productCatalog, sampleBookId);

        // Couple of sample updates.
```

```
            UpdateMultipleAttributes(productCatalog, sampleBookId);
            UpdateBookPriceConditionally(productCatalog, sampleBookId);

            // Delete.
            await DeleteBook(productCatalog, sampleBookId);
        }

        /// <summary>
        /// Loads the contents of a DynamoDB table.
        /// </summary>
        /// <param name="client">An initialized DynamoDB client object.</param>
        /// <param name="tableName">The name of the table to load.</param>
        /// <returns>A DynamoDB table object.</returns>
        public static Table LoadTable(IAmazonDynamoDB client, string tableName)
        {
            Table productCatalog = Table.LoadTable(client, tableName);
            return productCatalog;
        }

        /// <summary>
        /// Creates an example book item and adds it to the DynamoDB table
        /// ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async Task CreateBookItem(Table productCatalog, int
sampleBookId)
        {
            Console.WriteLine("\n*** Executing CreateBookItem() ***");
            var book = new Document
            {
                ["Id"] = sampleBookId,
                ["Title"] = "Book " + sampleBookId,
                ["Price"] = 19.99,
                ["ISBN"] = "111-1111111111",
                ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
                ["PageCount"] = 500,
                ["Dimensions"] = "8.5x11x.5",
                ["InPublication"] = new DynamoDBBool(true),
                ["InStock"] = new DynamoDBBool(false),
                ["QuantityOnHand"] = 0,
            };
```

```
            // Adds the book to the ProductCatalog table.
            await productCatalog.PutItemAsync(book);
        }

        /// <summary>
        /// Retrieves an item, a book, from the DynamoDB ProductCatalog table.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void RetrieveBook(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing RetrieveBook() ***");

            // Optional configuration.
            var config = new GetItemOperationConfig
            {
                AttributesToGet = new List<string> { "Id", "ISBN", "Title",
"Authors", "Price" },
                ConsistentRead = true,
            };

            Document document = await productCatalog.GetItemAsync(sampleBookId,
config);
            Console.WriteLine("RetrieveBook: Printing book retrieved...");
            PrintDocument(document);
        }

        /// <summary>
        /// Updates multiple attributes for a book and writes the changes to the
        /// DynamoDB table ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void UpdateMultipleAttributes(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\nUpdating multiple attributes....");
            int partitionKey = sampleBookId;
```

```
            var book = new Document
            {
                ["Id"] = partitionKey,

                // List of attribute updates.
                // The following replaces the existing authors list.
                ["Authors"] = new List<string> { "Author x", "Author y" },
                ["newAttribute"] = "New Value",
                ["ISBN"] = null, // Remove it.
            };

            // Optional parameters.
            var config = new UpdateItemOperationConfig
            {
                // Gets updated item in response.
                ReturnValues = ReturnValues.AllNewAttributes,
            };

            Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
            Console.WriteLine("UpdateMultipleAttributes: Printing item after
updates ...");
            PrintDocument(updatedBook);
        }

        /// <summary>
        /// Updates a book item if it meets the specified criteria.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async void UpdateBookPriceConditionally(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing UpdateBookPriceConditionally()
***");

            int partitionKey = sampleBookId;

            var book = new Document
            {
                ["Id"] = partitionKey,
```

```
                ["Price"] = 29.99,
            };

            // For conditional price update, creating a condition expression.
            var expr = new Expression
            {
                ExpressionStatement = "Price = :val",
            };
            expr.ExpressionAttributeValues[":val"] = 19.00;

            // Optional parameters.
            var config = new UpdateItemOperationConfig
            {
                ConditionalExpression = expr,
                ReturnValues = ReturnValues.AllNewAttributes,
            };

            Document updatedBook = await productCatalog.UpdateItemAsync(book,
config);
            Console.WriteLine("UpdateBookPriceConditionally: Printing item whose
price was conditionally updated");
            PrintDocument(updatedBook);
        }

        /// <summary>
        /// Deletes the book with the supplied Id value from the DynamoDB table
        /// ProductCatalog.
        /// </summary>
        /// <param name="productCatalog">A DynamoDB table object.</param>
        /// <param name="sampleBookId">An integer value representing the book's
ID.</param>
        public static async Task DeleteBook(
          Table productCatalog,
          int sampleBookId)
        {
            Console.WriteLine("\n*** Executing DeleteBook() ***");

            // Optional configuration.
            var config = new DeleteItemOperationConfig
            {
                // Returns the deleted item.
                ReturnValues = ReturnValues.AllOldAttributes,
            };
```

```
            Document document = await
productCatalog.DeleteItemAsync(sampleBookId, config);
            Console.WriteLine("DeleteBook: Printing deleted just deleted...");

            PrintDocument(document);
        }

        /// <summary>
        /// Prints the information for the supplied DynamoDB document.
        /// </summary>
        /// <param name="updatedDocument">A DynamoDB document object.</param>
        public static void PrintDocument(Document updatedDocument)
        {
            if (updatedDocument is null)
            {
                return;
            }

            foreach (var attribute in updatedDocument.GetAttributeNames())
            {
                string stringValue = null;
                var value = updatedDocument[attribute];

                if (value is null)
                {
                    continue;
                }

                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}", attribute,
stringValue);
            }
        }
```

```
    }
```

Realice operaciones de escritura por lotes con un modelo de documento.

```
    /// <summary>
    /// Shows how to use mid-level Amazon DynamoDB API calls to perform batch
    /// operations.
    /// </summary>
    public class MidLevelBatchWriteItem
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            await SingleTableBatchWrite(client);
            await MultiTableBatchWrite(client);
        }

        /// <summary>
        /// Perform a batch operation on a single DynamoDB table.
        /// </summary>
        /// <param name="client">An initialized DynamoDB object.</param>
        public static async Task SingleTableBatchWrite(IAmazonDynamoDB client)
        {
            Table productCatalog = Table.LoadTable(client, "ProductCatalog");
            var batchWrite = productCatalog.CreateBatchWrite();

            var book1 = new Document
            {
                ["Id"] = 902,
                ["Title"] = "My book1 in batch write using .NET helper classes",
                ["ISBN"] = "902-11-11-1111",
                ["Price"] = 10,
                ["ProductCategory"] = "Book",
                ["Authors"] = new List<string> { "Author 1", "Author 2", "Author
3" },
                ["Dimensions"] = "8.5x11x.5",
                ["InStock"] = new DynamoDBBool(true),
                ["QuantityOnHand"] = new DynamoDBNull(), // Quantity is unknown
at this time.
```

```
            };

            batchWrite.AddDocumentToPut(book1);

            // Specify delete item using overload that takes PK.
            batchWrite.AddKeyToDelete(12345);
            Console.WriteLine("Performing batch write in
SingleTableBatchWrite()");
            await batchWrite.ExecuteAsync();
        }

        /// <summary>
        /// Perform a batch operation involving multiple DynamoDB tables.
        /// </summary>
        /// <param name="client">An initialized DynamoDB client object.</param>
        public static async Task MultiTableBatchWrite(IAmazonDynamoDB client)
        {
            // Specify item to add in the Forum table.
            Table forum = Table.LoadTable(client, "Forum");
            var forumBatchWrite = forum.CreateBatchWrite();

            var forum1 = new Document
            {
                ["Name"] = "Test BatchWrite Forum",
                ["Threads"] = 0,
            };
            forumBatchWrite.AddDocumentToPut(forum1);

            // Specify item to add in the Thread table.
            Table thread = Table.LoadTable(client, "Thread");
            var threadBatchWrite = thread.CreateBatchWrite();

            var thread1 = new Document
            {
                ["ForumName"] = "S3 forum",
                ["Subject"] = "My sample question",
                ["Message"] = "Message text",
                ["KeywordTags"] = new List<string> { "S3", "Bucket" },
            };
            threadBatchWrite.AddDocumentToPut(thread1);

            // Specify item to delete from the Thread table.
            threadBatchWrite.AddKeyToDelete("someForumName", "someSubject");
```

```
            // Create multi-table batch.
            var superBatch = new MultiTableDocumentBatchWrite();
            superBatch.AddBatch(forumBatchWrite);
            superBatch.AddBatch(threadBatchWrite);
            Console.WriteLine("Performing batch write in
MultiTableBatchWrite()");

            // Execute the batch.
            await superBatch.ExecuteAsync();
        }
    }
```

Examine una tabla con un modelo de documento.

```
    /// <summary>
    /// Shows how to use mid-level Amazon DynamoDB API calls to scan a DynamoDB
    /// table for values.
    /// </summary>
    public class MidLevelScanOnly
    {
        public static async Task Main()
        {
            IAmazonDynamoDB client = new AmazonDynamoDBClient();

            Table productCatalogTable = Table.LoadTable(client,
"ProductCatalog");

            await FindProductsWithNegativePrice(productCatalogTable);
            await FindProductsWithNegativePriceWithConfig(productCatalogTable);
        }

        /// <summary>
        /// Retrieves any products that have a negative price in a DynamoDB
table.
        /// </summary>
        /// <param name="productCatalogTable">A DynamoDB table object.</param>
        public static async Task FindProductsWithNegativePrice(
          Table productCatalogTable)
        {
```

```
            // Assume there is a price error. So we scan to find items priced <
0.
            var scanFilter = new ScanFilter();
            scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

            Search search = productCatalogTable.Scan(scanFilter);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindProductsWithNegativePrice:
printing .............");

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Finds any items in the ProductCatalog table using a DynamoDB
        /// configuration object.
        /// </summary>
        /// <param name="productCatalogTable">A DynamoDB table object.</param>
        public static async Task FindProductsWithNegativePriceWithConfig(
          Table productCatalogTable)
        {
            // Assume there is a price error. So we scan to find items priced <
0.
            var scanFilter = new ScanFilter();
            scanFilter.AddCondition("Price", ScanOperator.LessThan, 0);

            var config = new ScanOperationConfig()
            {
                Filter = scanFilter,
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string> { "Title", "Id" },
            };

            Search search = productCatalogTable.Scan(config);

            do
```

```
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindProductsWithNegativePriceWithConfig:
printing .....");

                foreach (var document in documentList)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Displays the details of the passed DynamoDB document object on the
        /// console.
        /// </summary>
        /// <param name="document">A DynamoDB document object.</param>
        public static void PrintDocument(Document document)
        {
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
                var value = document[attribute];
                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}");
            }
        }
    }
```

Consulte y examine una tabla con un modelo de documento.

```
   /// <summary>
   /// Shows how to perform mid-level query procedures on an Amazon DynamoDB
   /// table.
   /// </summary>
   public class MidLevelQueryAndScan
   {
       public static async Task Main()
       {
           IAmazonDynamoDB client = new AmazonDynamoDBClient();

           // Query examples.
           Table replyTable = Table.LoadTable(client, "Reply");
           string forumName = "Amazon DynamoDB";
           string threadSubject = "DynamoDB Thread 2";

           await FindRepliesInLast15Days(replyTable);
           await FindRepliesInLast15DaysWithConfig(replyTable, forumName,
threadSubject);
           await FindRepliesPostedWithinTimePeriod(replyTable, forumName,
threadSubject);

           // Get Example.
           Table productCatalogTable = Table.LoadTable(client,
"ProductCatalog");
           int productId = 101;

           await GetProduct(productCatalogTable, productId);
       }

       /// <summary>
       /// Retrieves information about a product from the DynamoDB table
       /// ProductCatalog based on the product ID and displays the information
       /// on the console.
       /// </summary>
       /// <param name="tableName">The name of the table from which to retrieve
       /// product information.</param>
       /// <param name="productId">The ID of the product to retrieve.</param>
       public static async Task GetProduct(Table tableName, int productId)
       {
           Console.WriteLine("*** Executing GetProduct() ***");
           Document productDocument = await tableName.GetItemAsync(productId);
```

```
            if (productDocument != null)
            {
                PrintDocument(productDocument);
            }
            else
            {
                Console.WriteLine("Error: product " + productId + " does not
exist");
            }
        }

        /// <summary>
        /// Retrieves replies from the passed DynamoDB table object.
        /// </summary>
        /// <param name="table">The table we want to query.</param>
        public static async Task FindRepliesInLast15Days(
          Table table)
        {
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            var filter = new QueryFilter("Id", QueryOperator.Equal, "Id");
            filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

            // Use Query overloads that take the minimum required query
parameters.
            Search search = table.Query(filter);

            do
            {
                var documentSet = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesInLast15Days:
printing .....");

                foreach (var document in documentSet)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Retrieve replies made during a specific time period.
        /// </summary>
```

```
        /// <param name="table">The table we want to query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadSubject">The subject of the thread, which we are
        /// searching for replies.</param>
        public static async Task FindRepliesPostedWithinTimePeriod(
          Table table,
          string forumName,
          string threadSubject)
        {
            DateTime startDate = DateTime.UtcNow.Subtract(new TimeSpan(21, 0, 0,
0));
            DateTime endDate = DateTime.UtcNow.Subtract(new TimeSpan(1, 0, 0,
0));

            var filter = new QueryFilter("Id", QueryOperator.Equal, forumName +
"#" + threadSubject);
            filter.AddCondition("ReplyDateTime", QueryOperator.Between,
startDate, endDate);

            var config = new QueryOperationConfig()
            {
                Limit = 2, // 2 items/page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
        {
          "Message",
          "ReplyDateTime",
          "PostedBy",
        },
                ConsistentRead = true,
                Filter = filter,
            };

            Search search = table.Query(config);

            do
            {
                var documentList = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesPostedWithinTimePeriod: printing
replies posted within dates: {0} and {1} .....", startDate, endDate);

                foreach (var document in documentList)
                {
```

```
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Perform a query for replies made in the last 15 days using a DynamoDB
        /// QueryOperationConfig object.
        /// </summary>
        /// <param name="table">The table we want to query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadName">The bane of the thread that we are searching
        /// for replies.</param>
        public static async Task FindRepliesInLast15DaysWithConfig(
          Table table,
          string forumName,
          string threadName)
        {
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);
            var filter = new QueryFilter("Id", QueryOperator.Equal, forumName +
"#" + threadName);
            filter.AddCondition("ReplyDateTime", QueryOperator.GreaterThan,
twoWeeksAgoDate);

            var config = new QueryOperationConfig()
            {
                Filter = filter,

                // Optional parameters.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                  "Message",
                  "ReplyDateTime",
                  "PostedBy",
                },
                ConsistentRead = true,
            };

            Search search = table.Query(config);

            do
```

```
            {
                var documentSet = await search.GetNextSetAsync();
                Console.WriteLine("\nFindRepliesInLast15DaysWithConfig:
printing .............");

                foreach (var document in documentSet)
                {
                    PrintDocument(document);
                }
            }
            while (!search.IsDone);
        }

        /// <summary>
        /// Displays the contents of the passed DynamoDB document on the console.
        /// </summary>
        /// <param name="document">A DynamoDB document to display.</param>
        public static void PrintDocument(Document document)
        {
            Console.WriteLine();
            foreach (var attribute in document.GetAttributeNames())
            {
                string stringValue = null;
                var value = document[attribute];

                if (value is Primitive)
                {
                    stringValue = value.AsPrimitive().Value.ToString();
                }
                else if (value is PrimitiveList)
                {
                    stringValue = string.Join(",", (from primitive
                      in value.AsPrimitiveList().Entries
                                                    select
primitive.Value).ToArray());
                }

                Console.WriteLine($"{attribute} - {stringValue}");
            }
        }
    }
```

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Utilizar un modelo de persistencia de objetos de alto nivel para DynamoDB mediante un AWS SDK

En el siguiente ejemplo de código se muestra cómo realizar operaciones de creación, lectura, actualización y eliminación (CRUD) y por lotes mediante un modelo de persistencia de objetos para DynamoDB y un AWS SDK.

Para obtener más información, consulte Object persistence model (Modelo de persistencia de objetos).

.NET

SDK para .NET

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el Repositorio de ejemplos de código de AWS.

Realice operaciones de CRUD mediante un modelo de persistencia de objetos de alto nivel.

```
/// <summary>
/// Shows how to perform high-level CRUD operations on an Amazon DynamoDB
/// table.
/// </summary>
public class HighLevelItemCrud
{
    public static async Task Main()
    {
        var client = new AmazonDynamoDBClient();
        DynamoDBContext context = new DynamoDBContext(client);
        await PerformCRUDOperations(context);
    }

    public static async Task PerformCRUDOperations(IDynamoDBContext context)
```

```
        {
            int bookId = 1001; // Some unique value.
            Book myBook = new Book
            {
                Id = bookId,
                Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
                Isbn = "111-1111111001",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
            };

            // Save the book to the ProductCatalog table.
            await context.SaveAsync(myBook);

            // Retrieve the book from the ProductCatalog table.
            Book bookRetrieved = await context.LoadAsync<Book>(bookId);

            // Update some properties.
            bookRetrieved.Isbn = "222-2222221001";

            // Update existing authors list with the following values.
            bookRetrieved.BookAuthors = new List<string> { " Author 1", "Author
x" };
            await context.SaveAsync(bookRetrieved);

            // Retrieve the updated book. This time, add the optional
            // ConsistentRead parameter using DynamoDBContextConfig object.
            await context.LoadAsync<Book>(bookId, new DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            // Delete the book.
            await context.DeleteAsync<Book>(bookId);

            // Try to retrieve deleted book. It should return null.
            Book deletedBook = await context.LoadAsync<Book>(bookId, new
DynamoDBContextConfig
            {
                ConsistentRead = true,
            });

            if (deletedBook == null)
            {
                Console.WriteLine("Book is deleted");
```

```
        }
    }
}
```

Realice operaciones de escritura por lotes mediante un modelo de persistencia de objetos de alto nivel.

```
/// <summary>
/// Performs high-level batch write operations to an Amazon DynamoDB table.
/// This example was written using the AWS SDK for .NET version 3.7 and .NET
/// Core 5.0.
/// </summary>
public class HighLevelBatchWriteItem
{
    public static async Task SingleTableBatchWrite(IDynamoDBContext context)
    {
        Book book1 = new Book
        {
            Id = 902,
            InPublication = true,
            Isbn = "902-11-11-1111",
            PageCount = "100",
            Price = 10,
            ProductCategory = "Book",
            Title = "My book3 in batch write",
        };

        Book book2 = new Book
        {
            Id = 903,
            InPublication = true,
            Isbn = "903-11-11-1111",
            PageCount = "200",
            Price = 10,
            ProductCategory = "Book",
            Title = "My book4 in batch write",
        };

        var bookBatch = context.CreateBatchWrite<Book>();
        bookBatch.AddPutItems(new List<Book> { book1, book2 });
```

```
            Console.WriteLine("Adding two books to ProductCatalog table.");
            await bookBatch.ExecuteAsync();
        }

        public static async Task MultiTableBatchWrite(IDynamoDBContext context)
        {
            // New Forum item.
            Forum newForum = new Forum
            {
                Name = "Test BatchWrite Forum",
                Threads = 0,
            };
            var forumBatch = context.CreateBatchWrite<Forum>();
            forumBatch.AddPutItem(newForum);

            // New Thread item.
            Thread newThread = new Thread
            {
                ForumName = "S3 forum",
                Subject = "My sample question",
                KeywordTags = new List<string> { "S3", "Bucket" },
                Message = "Message text",
            };

            DynamoDBOperationConfig config = new DynamoDBOperationConfig();
            config.SkipVersionCheck = true;
            var threadBatch = context.CreateBatchWrite<Thread>(config);
            threadBatch.AddPutItem(newThread);
            threadBatch.AddDeleteKey("some partition key value", "some sort key
value");

            var superBatch = new MultiTableBatchWrite(forumBatch, threadBatch);

            Console.WriteLine("Performing batch write in
MultiTableBatchWrite().");
            await superBatch.ExecuteAsync();
        }

        public static async Task Main()
        {
            AmazonDynamoDBClient client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);
```

```
            await SingleTableBatchWrite(context);
            await MultiTableBatchWrite(context);
        }
    }
```

Mapee datos arbitrarios a una tabla mediante un modelo de persistencia de objetos de alto nivel.

```
    /// <summary>
    /// Shows how to map arbitrary data to an Amazon DynamoDB table.
    /// </summary>
    public class HighLevelMappingArbitraryData
    {
        /// <summary>
        /// Creates a book, adds it to the DynamoDB ProductCatalog table,
retrieves
        /// the new book from the table, updates the dimensions and writes the
        /// changed item back to the table.
        /// </summary>
        /// <param name="context">The DynamoDB context object used to write and
        /// read data from the table.</param>
        public static async Task AddRetrieveUpdateBook(IDynamoDBContext context)
        {
            // Create a book.
            DimensionType myBookDimensions = new DimensionType()
            {
                Length = 8M,
                Height = 11M,
                Thickness = 0.5M,
            };

            Book myBook = new Book
            {
                Id = 501,
                Title = "AWS SDK for .NET Object Persistence Model Handling
Arbitrary Data",
                Isbn = "999-9999999999",
                BookAuthors = new List<string> { "Author 1", "Author 2" },
                Dimensions = myBookDimensions,
            };
```

```
            // Add the book to the DynamoDB table ProductCatalog.
            await context.SaveAsync(myBook);

            // Retrieve the book.
            Book bookRetrieved = await context.LoadAsync<Book>(501);

            // Update the book dimensions property.
            bookRetrieved.Dimensions.Height += 1;
            bookRetrieved.Dimensions.Length += 1;
            bookRetrieved.Dimensions.Thickness += 0.2M;

            // Write the changed item to the table.
            await context.SaveAsync(bookRetrieved);
        }

        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();
            DynamoDBContext context = new DynamoDBContext(client);
            await AddRetrieveUpdateBook(context);
        }
    }
```

Consulte y examine una tabla mediante un modelo de persistencia de objetos de alto nivel.

```
    /// <summary>
    /// Shows how to perform high-level query and scan operations to Amazon
    /// DynamoDB tables.
    /// </summary>
    public class HighLevelQueryAndScan
    {
        public static async Task Main()
        {
            var client = new AmazonDynamoDBClient();

            DynamoDBContext context = new DynamoDBContext(client);

            // Get an item.
            await GetBook(context, 101);
```

```
            // Sample forum and thread to test queries.
            string forumName = "Amazon DynamoDB";
            string threadSubject = "DynamoDB Thread 1";

            // Sample queries.
            await FindRepliesInLast15Days(context, forumName, threadSubject);
            await FindRepliesPostedWithinTimePeriod(context, forumName,
threadSubject);

            // Scan table.
            await FindProductsPricedLessThanZero(context);
        }

        public static async Task GetBook(IDynamoDBContext context, int productId)
        {
            Book bookItem = await context.LoadAsync<Book>(productId);

            Console.WriteLine("\nGetBook: Printing result.....");
            Console.WriteLine($"Title: {bookItem.Title} \n ISBN:{bookItem.Isbn}
\n No. of pages: {bookItem.PageCount}");
        }

        /// <summary>
        /// Queries a DynamoDB table to find replies posted within the last 15
days.
        /// </summary>
        /// <param name="context">The DynamoDB context used to perform the
query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadSubject">The thread object containing the query
parameters.</param>
        public static async Task FindRepliesInLast15Days(
          IDynamoDBContext context,
          string forumName,
          string threadSubject)
        {
            string replyId = $"{forumName} #{threadSubject}";
            DateTime twoWeeksAgoDate = DateTime.UtcNow - TimeSpan.FromDays(15);

            List<object> times = new List<object>();
            times.Add(twoWeeksAgoDate);
```

```
            List<ScanCondition> scs = new List<ScanCondition>();
            var sc = new ScanCondition("PostedBy", ScanOperator.GreaterThan,
times.ToArray());
            scs.Add(sc);

            var cfg = new DynamoDBOperationConfig
            {
                QueryFilter = scs,
            };

            AsyncSearch<Reply> response = context.QueryAsync<Reply>(replyId,
cfg);
            IEnumerable<Reply> latestReplies = await
response.GetRemainingAsync();

            Console.WriteLine("\nReplies in last 15 days:");

            foreach (Reply r in latestReplies)
            {

Console.WriteLine($"{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
            }
        }

        /// <summary>
        /// Queries for replies posted within a specific time period.
        /// </summary>
        /// <param name="context">The DynamoDB context used to perform the
query.</param>
        /// <param name="forumName">The name of the forum that we're interested
in.</param>
        /// <param name="threadSubject">Information about the subject that we're
        /// interested in.</param>
        public static async Task FindRepliesPostedWithinTimePeriod(
          IDynamoDBContext context,
          string forumName,
          string threadSubject)
        {
            string forumId = forumName + "#" + threadSubject;
            Console.WriteLine("\nReplies posted within time period:");

            DateTime startDate = DateTime.UtcNow - TimeSpan.FromDays(30);
            DateTime endDate = DateTime.UtcNow - TimeSpan.FromDays(1);
```

```
            List<object> times = new List<object>();
            times.Add(startDate);
            times.Add(endDate);

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc = new ScanCondition("LastPostedBy", ScanOperator.Between,
times.ToArray());
            scs.Add(sc);

            var cfg = new DynamoDBOperationConfig
            {
                QueryFilter = scs,
            };

            AsyncSearch<Reply> response = context.QueryAsync<Reply>(forumId,
cfg);
            IEnumerable<Reply> repliesInAPeriod = await
response.GetRemainingAsync();

            foreach (Reply r in repliesInAPeriod)
            {

Console.WriteLine("{r.Id}\t{r.PostedBy}\t{r.Message}\t{r.ReplyDateTime}");
            }
        }

        /// <summary>
        /// Queries the DynamoDB ProductCatalog table for products costing less
        /// than zero.
        /// </summary>
        /// <param name="context">The DynamoDB context object used to perform the
        /// query.</param>
        public static async Task FindProductsPricedLessThanZero(IDynamoDBContext
context)
        {
            int price = 0;

            List<ScanCondition> scs = new List<ScanCondition>();
            var sc1 = new ScanCondition("Price", ScanOperator.LessThan, price);
            var sc2 = new ScanCondition("ProductCategory", ScanOperator.Equal,
"Book");
            scs.Add(sc1);
            scs.Add(sc2);
```

```
            AsyncSearch<Book> response = context.ScanAsync<Book>(scs);

            IEnumerable<Book> itemsWithWrongPrice = await
response.GetRemainingAsync();

            Console.WriteLine("\nFindProductsPricedLessThanZero: Printing
result.....");

            foreach (Book r in itemsWithWrongPrice)
            {
                Console.WriteLine($"{r.Id}\t{r.Title}\t{r.Price}\t{r.Isbn}");
            }
        }
    }
```

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de operaciones de contador atómico en DynamoDB con un AWS SDK

En los siguientes ejemplos de código se muestra cómo utilizar operaciones de contador atómico en DynamoDB.

- Incremente contadores de forma atómica con operaciones ADD y SET.
- Incremente de forma segura contadores que podrían no existir.
- Implementar bloqueo positivo para operaciones de contador.

Java

SDK para Java 2.x

Demuestre las operaciones de contador atómico con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
```

```
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;

    /**
     * Increments a counter using the ADD operation.
     *
     * <p>This method demonstrates how to use the ADD operation to atomically
     * increment a counter attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param incrementValue The value to increment by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse incrementCounterWithAdd(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int incrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("ADD #counterName :increment")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":increment",

 AttributeValue.builder().n(String.valueOf(incrementValue)).build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
```

```
    }

    /**
     * Increments a counter using the SET operation.
     *
     * <p>This method demonstrates how to use the SET operation with an
expression
     * to increment a counter attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param incrementValue The value to increment by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse incrementCounterWithSet(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int incrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #counterName = #counterName + :increment")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":increment",

AttributeValue.builder().n(String.valueOf(incrementValue)).build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Increments a counter safely, handling the case where the counter doesn't
exist yet.
```

```
     *
     * <p>This method demonstrates how to use if_not_exists to safely increment a
counter
     * that may not exist yet.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param incrementValue The value to increment by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse incrementCounterSafely(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int incrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #counterName =
if_not_exists(#counterName, :zero) + :increment")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":increment",

AttributeValue.builder().n(String.valueOf(incrementValue)).build(),
                ":zero", AttributeValue.builder().n("0").build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Decrements a counter safely, ensuring it doesn't go below zero.
     *
     * <p>This method demonstrates how to use a condition expression to safely
     * decrement a counter without going below zero.
```

```
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param counterName The name of the counter attribute
     * @param decrementValue The value to decrement by
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation or if
the counter would go below zero
     */
    public static UpdateItemResponse decrementCounterSafely(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String counterName,
        int decrementValue) {

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #counterName = #counterName - :decrement")
            .conditionExpression("#counterName >= :decrement")
            .expressionAttributeNames(Map.of("#counterName", counterName))
            .expressionAttributeValues(Map.of(
                ":decrement",

AttributeValue.builder().n(String.valueOf(decrementValue)).build()))
            .returnValues(ReturnValue.UPDATED_NEW)
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Compares the ADD and SET approaches for incrementing counters.
     *
     * <p>This method demonstrates the differences between using ADD and SET
     * for incrementing counters in DynamoDB.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
```

```
     * @return Map containing the comparison results
     */
    public static Map<String, Object> compareAddVsSet(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key) {

        Map<String, Object> results = new HashMap<>();

        try {
            // Reset counters to ensure a fair comparison
            UpdateItemRequest resetRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET AddCounter = :zero, SetCounter = :zero")
                .expressionAttributeValues(
                    Map.of(":zero", AttributeValue.builder().n("0").build()))
                .build();

            dynamoDbClient.updateItem(resetRequest);

            // Increment with ADD
            long addStartTime = System.nanoTime();
            UpdateItemResponse addResponse =
incrementCounterWithAdd(dynamoDbClient, tableName, key, "AddCounter", 1);
            long addEndTime = System.nanoTime();
            long addDuration = addEndTime - addStartTime;

            // Increment with SET
            long setStartTime = System.nanoTime();
            UpdateItemResponse setResponse =
incrementCounterWithSet(dynamoDbClient, tableName, key, "SetCounter", 1);
            long setEndTime = System.nanoTime();
            long setDuration = setEndTime - setStartTime;

            // Record results
            results.put("addResponse", addResponse);
            results.put("setResponse", setResponse);
            results.put("addDuration", addDuration);
            results.put("setDuration", setDuration);
            results.put("success", true);

        } catch (DynamoDbException e) {
            results.put("success", false);
            results.put("error", e.getMessage());
```

```
        }

        return results;
    }

    /**
     * Gets the current value of a counter attribute.
     *
     * <p>Helper method to retrieve the current value of a counter attribute.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to get
     * @param counterName The name of the counter attribute
     * @return The counter value or null if not found
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static Integer getCounterValue(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String counterName) {

        // Define the get parameters
        GetItemRequest request = GetItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .projectionExpression(counterName)
            .build();

        // Perform the get operation
        GetItemResponse response = dynamoDbClient.getItem(request);

        // Return the counter value if it exists, otherwise null
        if (response.item() != null && response.item().containsKey(counterName))
{
            return Integer.parseInt(response.item().get(counterName).n());
        }

        return null;
    }
```

Ejemplo de uso de operaciones de contador atómico con AWS SDK for Java 2.x.

```
   public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
       // Example key
       Map<String, AttributeValue> key = new HashMap<>();
       key.put("ProductId", AttributeValue.builder().s("P12345").build());

       System.out.println("Demonstrating atomic counter operations in
DynamoDB");

       try {
           // Example 1: Increment a counter using ADD
           System.out.println("\nExample 1: Incrementing a counter using ADD");
           UpdateItemResponse addResponse =
incrementCounterWithAdd(dynamoDbClient, tableName, key, "ViewCount", 1);

           System.out.println("Updated counter: " + addResponse.attributes());

           // Example 2: Increment a counter using SET
           System.out.println("\nExample 2: Incrementing a counter using SET");
           UpdateItemResponse setResponse =
incrementCounterWithSet(dynamoDbClient, tableName, key, "LikeCount", 1);

           System.out.println("Updated counter: " + setResponse.attributes());

           // Example 3: Increment a counter safely
           System.out.println("\nExample 3: Incrementing a counter safely");
           UpdateItemResponse safeResponse =
incrementCounterSafely(dynamoDbClient, tableName, key, "ShareCount", 1);

           System.out.println("Updated counter: " + safeResponse.attributes());

           // Example 4: Decrement a counter safely
           System.out.println("\nExample 4: Decrementing a counter safely");
           try {
               UpdateItemResponse decrementResponse =
                   decrementCounterSafely(dynamoDbClient, tableName, key,
"InventoryCount", 1);

               System.out.println("Updated counter: " +
decrementResponse.attributes());
           } catch (DynamoDbException e) {
               if (e.getMessage().contains("ConditionalCheckFailed")) {
                   System.out.println("Cannot decrement counter below zero");
```

```
                } else {
                    throw e;
                }
            }

            // Example 5: Compare ADD vs SET
            System.out.println("\nExample 5: Comparing ADD vs SET");
            Map<String, Object> comparison = compareAddVsSet(dynamoDbClient,
tableName, key);

            if ((boolean) comparison.get("success")) {
                System.out.println("ADD duration: " +
comparison.get("addDuration") + " ns");
                System.out.println("SET duration: " +
comparison.get("setDuration") + " ns");
                System.out.println("ADD response: " +
comparison.get("addResponse"));
                System.out.println("SET response: " +
comparison.get("setResponse"));
            } else {
                System.out.println("Comparison failed: " +
comparison.get("error"));
            }

            // Explain atomic counter operations
            System.out.println("\nKey points about DynamoDB atomic counter
operations:");
            System.out.println("1. Both ADD and SET can be used for atomic
counters");
            System.out.println("2. ADD is more concise for simple increments");
            System.out.println("3. SET with an expression is more flexible for
complex operations");
            System.out.println("4. Use if_not_exists to handle the case where the
counter doesn't exist yet");
            System.out.println("5. Use condition expressions to prevent counters
from going below zero");
            System.out.println("6. Atomic operations are guaranteed to be
isolated from other writes");
            System.out.println("7. ADD can only be used with number and set data
types");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
```

```
        }
    }
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK for Java 2.x.

JavaScript

SDK para JavaScript (v3)

Demuestre las operaciones de contador atómico con AWS SDK para JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Increment a counter using the ADD operation.
 *
 * This function demonstrates using the ADD operation for atomic increments.
 * The ADD operation is atomic and is the recommended way to increment counters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithAdd(
  config,
  tableName,
  key,
  counterName,
  incrementValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Define the update parameters using ADD
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `ADD ${counterName} :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter using the SET operation with an expression.
 *
 * This function demonstrates using the SET operation with an expression for
 increments.
 * While this approach works, it's less idiomatic for simple increments than
 using ADD.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithSet(
  config,
  tableName,
  key,
  counterName,
  incrementValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Define the update parameters using SET with an expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter safely, handling the case where the counter might not
 exist.
 *
 * This function demonstrates using the if_not_exists function with SET to safely
 * increment a counter that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} defaultValue - The default value if the counter doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterSafely(
  config,
  tableName,
  key,
  counterName,
  incrementValue,
  defaultValue = 0
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);
```

```
  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} =
 if_not_exists(${counterName}, :default) + :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue,
      ":default": defaultValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Increment a counter with optimistic locking to prevent race conditions.
 *
 * This function demonstrates using a condition expression to implement
 optimistic
 * locking, which prevents race conditions when multiple processes try to update
 * the same counter.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} expectedValue - The expected current value of the counter
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithLocking(
  config,
  tableName,
  key,
  counterName,
  incrementValue,
  expectedValue
) {
  // Initialize the DynamoDB client
```

```
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
    ConditionExpression: `${counterName} = :expected`,
    ExpressionAttributeValues: {
      ":increment": incrementValue,
      ":expected": expectedValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return {
      success: true,
      data: response
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        error: "Optimistic locking failed: the counter value has changed"
      };
    }
    // Re-throw other errors
    throw error;
  }
}

/**
 * Get the current value of a counter.
 *
 * Helper function to retrieve the current value of a counter attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
```

```
 * @param {string} counterName - The name of the counter attribute
 * @returns {Promise<number|null>} - The current counter value or null if not
 found
 */
async function getCounterValue(
  config,
  tableName,
  key,
  counterName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the counter value if it exists, otherwise null
  return response.Item && counterName in response.Item
    ? response.Item[counterName]
    : null;
}
```

Ejemplo de uso de operaciones de contador atómico con AWS SDK para JavaScript.

```
/**
 * Example of how to use the atomic counter operations.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };
  const counterName = "ViewCount";
  const incrementValue = 1;
```

```
 console.log("Demonstrating different approaches to increment counters in
DynamoDB");

 try {
   // Example 1: Using ADD operation (recommended for simple increments)
   console.log("\nExample 1: Incrementing counter with ADD operation");
   const response1 = await incrementCounterWithAdd(
     config,
     tableName,
     key,
     counterName,
     incrementValue
   );

   console.log(`Counter incremented to: ${response1.Attributes[counterName]}`);

   // Example 2: Using SET operation with an expression
   console.log("\nExample 2: Incrementing counter with SET operation");
   const response2 = await incrementCounterWithSet(
     config,
     tableName,
     key,
     counterName,
     incrementValue
   );

   console.log(`Counter incremented to: ${response2.Attributes[counterName]}`);

   // Example 3: Safely incrementing a counter that might not exist
   console.log("\nExample 3: Safely incrementing counter that might not exist");
   const newKey = { ProductId: "P67890" };
   const response3 = await incrementCounterSafely(
     config,
     tableName,
     newKey,
     counterName,
     incrementValue,
     0
   );

   console.log(`Counter initialized and incremented to:
${response3.Attributes[counterName]}`);

   // Example 4: Incrementing with optimistic locking
```

```
    console.log("\nExample 4: Incrementing with optimistic locking");

    // First, get the current counter value
    const currentValue = await getCounterValue(config, tableName, key,
 counterName);
    console.log(`Current counter value: ${currentValue}`);

    // Then, try to increment with optimistic locking
    const response4 = await incrementCounterWithLocking(
      config,
      tableName,
      key,
      counterName,
      incrementValue,
      currentValue
    );

    if (response4.success) {
      console.log(`Counter successfully incremented to:
 ${response4.data.Attributes[counterName]}`);
    } else {
      console.log(response4.error);
    }

    // Explain the differences between ADD and SET
    console.log("\nKey differences between ADD and SET for counter operations:");
    console.log("1. ADD is more concise and idiomatic for simple increments");
    console.log("2. SET with expressions is more flexible for complex
 operations");
    console.log("3. Both operations are atomic and safe for concurrent updates");
    console.log("4. SET with if_not_exists is required when the attribute might
 not exist");
    console.log("5. Optimistic locking can be added to either approach for
 additional safety");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Para obtener información sobre la API, consulte UpdateItem en la referencia de la API de AWS SDK para JavaScript.

Python

SDK para Python (Boto3)

Demuestre las operaciones de contador atómico con AWS SDK for Python (Boto3).

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, Union


def increment_counter_with_add(
    table_name: str, key: Dict[str, Any], counter_name: str, increment_value: int
 = 1
) -> Dict[str, Any]:
    """
    Increment a counter attribute using the ADD operation.

    This function demonstrates the atomic ADD operation, which is ideal for
    incrementing counters without the risk of race conditions.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        increment_value (int, optional): The value to increment by. Defaults to
 1.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the ADD operation to atomically increment the counter
    response = table.update_item(
        Key=key,
        UpdateExpression="ADD #counter :increment",
        ExpressionAttributeNames={"#counter": counter_name},
        ExpressionAttributeValues={":increment": increment_value},
        ReturnValues="UPDATED_NEW",
    )
```

```
    return response


def increment_counter_with_set(
    table_name: str, key: Dict[str, Any], counter_name: str, increment_value: int
 = 1
) -> Dict[str, Any]:
    """
    Increment a counter attribute using the SET operation with an expression.

    This function demonstrates using SET with an expression to increment a
 counter.
    While this works, it's generally recommended to use ADD for simple
 increments.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        increment_value (int, optional): The value to increment by. Defaults to
 1.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use the SET operation with an expression to increment the counter
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #counter = #counter + :increment",
        ExpressionAttributeNames={"#counter": counter_name},
        ExpressionAttributeValues={":increment": increment_value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def increment_counter_safely(
```

```
    table_name: str,
    key: Dict[str, Any],
    counter_name: str,
    increment_value: int = 1,
    initial_value: int = 0,
) -> Dict[str, Any]:
    """
    Increment a counter attribute safely, handling the case where it might not
 exist.

    This function demonstrates a best practice for incrementing counters by using
    the if_not_exists function to handle the case where the counter doesn't exist
 yet.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        increment_value (int, optional): The value to increment by. Defaults to
 1.
        initial_value (int, optional): The initial value if the counter doesn't
 exist. Defaults to 0.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use SET with if_not_exists to safely increment the counter
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #counter = if_not_exists(#counter, :initial)
 + :increment",
        ExpressionAttributeNames={"#counter": counter_name},
        ExpressionAttributeValues={":increment": increment_value, ":initial":
 initial_value},
        ReturnValues="UPDATED_NEW",
    )

    return response
```

```
def atomic_conditional_increment(
    table_name: str,
    key: Dict[str, Any],
    counter_name: str,
    condition_attribute: str,
    condition_value: Any,
    increment_value: int = 1,
) -> Union[Dict[str, Any], None]:
    """
    Atomically increment a counter only if a condition is met.

    This function demonstrates combining atomic counter operations with
    conditional expressions for more complex update scenarios.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        counter_name (str): The name of the counter attribute.
        condition_attribute (str): The attribute to check in the condition.
        condition_value (Any): The value to compare against.
        increment_value (int, optional): The value to increment by. Defaults to
 1.

    Returns:
        Optional[Dict[str, Any]]: The response from DynamoDB if successful, None
 if condition failed.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        # Use ADD with a condition expression
        response = table.update_item(
            Key=key,
            UpdateExpression="ADD #counter :increment",
            ConditionExpression="#condition = :value",
            ExpressionAttributeNames={"#counter": counter_name, "#condition":
 condition_attribute},
            ExpressionAttributeValues={":increment": increment_value, ":value":
 condition_value},
            ReturnValues="UPDATED_NEW",
        )
```

```
        return response
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return None
        else:
            # Other error occurred
            raise
```

Ejemplo de uso de operaciones de contador atómico con AWS SDK for Python (Boto3).

```
def example_usage():
    """Example of how to use the atomic counter operations functions."""
    # Example parameters
    table_name = "GameScores"
    key = {"UserId": "user123", "GameId": "game456"}
    counter_name = "Score"

    print("Example 1: Incrementing a counter with ADD operation")
    try:
        response = increment_counter_with_add(
            table_name=table_name, key=key, counter_name=counter_name,
 increment_value=10
        )
        print(
            f"Counter incremented successfully. New value:
 {response.get('Attributes', {}).get(counter_name)}"
        )
    except Exception as e:
        print(f"Error incrementing counter with ADD: {e}")

    print("\nExample 2: Incrementing a counter with SET operation")
    try:
        response = increment_counter_with_set(
            table_name=table_name, key=key, counter_name=counter_name,
 increment_value=5
        )
        print(
            f"Counter incremented successfully. New value:
 {response.get('Attributes', {}).get(counter_name)}"
```

```
        )
    except Exception as e:
        print(f"Error incrementing counter with SET: {e}")

    print("\nExample 3: Safely incrementing a counter that might not exist")
    try:
        new_key = {"UserId": "newuser789", "GameId": "game456"}
        response = increment_counter_safely(
            table_name=table_name,
            key=new_key,
            counter_name=counter_name,
            increment_value=15,
            initial_value=100,
        )
        print(
            f"Counter safely incremented. New value: {response.get('Attributes',
{}).get(counter_name)}"
        )
    except Exception as e:
        print(f"Error safely incrementing counter: {e}")

    print("\nExample 4: Conditional counter increment")
    try:
        # Fix for mypy: Handle the case where response might be None
        result = atomic_conditional_increment(
            table_name=table_name,
            key=key,
            counter_name="Achievements",
            condition_attribute="Level",
            condition_value=5,
            increment_value=1,
        )

        if result is not None:
            print(
                f"Conditional increment succeeded. New value:
{result.get('Attributes', {}).get('Achievements')}"
            )
        else:
            print("Conditional increment failed because condition was not met.")
        if response:
            print(
                f"Conditional increment succeeded. New value:
{response.get('Attributes', {}).get('Achievements')}"
```

```
            )
        else:
            print("Conditional increment failed because condition was not met.")
    except Exception as e:
        print(f"Error with conditional increment: {e}")

    print("\nComparison of ADD vs SET for counter operations:")
    print("1. ADD is specifically designed for atomic numeric increments and set
operations")
    print("2. SET with an expression can be used for more complex calculations")
    print("3. Both operations are atomic, preventing race conditions")
    print("4. ADD is more concise for simple increments")
    print("5. SET with if_not_exists() is recommended when the attribute might
not exist")
    print("6. For counters, ADD is generally preferred for clarity and
simplicity")
```

- Para obtener información la API, consulte UpdateItem en la referencia de la API de AWS SDK para Python (Boto3).

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de operaciones condicionales en DynamoDB con un AWS SDK

En los siguientes ejemplos de código se muestra cómo utilizar operaciones condicionales en DynamoDB.

- Implemente escrituras condicionales para evitar la sobrescritura de datos.
- Utilice expresiones condicionales para aplicar reglas de negocio.
- Gestione los errores de comprobación condicionales de forma elegante.

Java

SDK para Java 2.x

Demuestre las operaciones condicionales con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import
 software.amazon.awssdk.services.dynamodb.model.ConditionalCheckFailedException;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemRequest;
import software.amazon.awssdk.services.dynamodb.model.DeleteItemResponse;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.GetItemResponse;
import software.amazon.awssdk.services.dynamodb.model.ReturnValue;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.HashMap;
import java.util.Map;

    /**
     * Performs a conditional update on an item.
     *
     * <p>This method demonstrates how to use a condition expression to update an
 item
     * only if a specific condition is met.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param conditionAttribute The attribute to check in the condition
     * @param conditionValue The value to compare against
     * @param updateAttribute The attribute to update
     * @param updateValue The new value to set
     * @return Map containing the operation result and status
     */
    public static Map<String, Object> conditionalUpdate(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String conditionAttribute,
        AttributeValue conditionValue,
```

```
        String updateAttribute,
        AttributeValue updateValue) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Define the update parameters
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #updateAttr = :updateVal")
                .conditionExpression("#condAttr = :condVal")
                .expressionAttributeNames(Map.of(
                    "#condAttr", conditionAttribute,
                    "#updateAttr", updateAttribute))
                .expressionAttributeValues(Map.of(
                    ":condVal", conditionValue,
                    ":updateVal", updateValue))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            // Perform the update operation
            UpdateItemResponse response = dynamoDbClient.updateItem(request);

            // Record success result
            result.put("success", true);
            result.put("message", "Condition was met and update was performed");
            result.put("attributes", response.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to condition not being met
            result.put("success", false);
            result.put("message", "Condition was not met, update was not
performed");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
```

```
    }

    /**
     * Performs a conditional delete on an item.
     *
     * <p>This method demonstrates how to use a condition expression to delete an
item
     * only if a specific condition is met.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to delete
     * @param conditionAttribute The attribute to check in the condition
     * @param conditionValue The value to compare against
     * @return Map containing the operation result and status
     */
    public static Map<String, Object> conditionalDelete(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String conditionAttribute,
        AttributeValue conditionValue) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Define the delete parameters
            DeleteItemRequest request = DeleteItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .conditionExpression("#condAttr = :condVal")
                .expressionAttributeNames(Map.of("#condAttr",
conditionAttribute))
                .expressionAttributeValues(Map.of(":condVal", conditionValue))
                .returnValues(ReturnValue.ALL_OLD)
                .build();

            // Perform the delete operation
            DeleteItemResponse response = dynamoDbClient.deleteItem(request);

            // Record success result
            result.put("success", true);
            result.put("message", "Condition was met and delete was performed");
            result.put("attributes", response.attributes());
```

```
        } catch (ConditionalCheckFailedException e) {
            // Record failure due to condition not being met
            result.put("success", false);
            result.put("message", "Condition was not met, delete was not
performed");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
    }

    /**
     * Demonstrates optimistic locking using a version attribute.
     *
     * <p>This method shows how to implement optimistic locking by using a
version
     * attribute that is incremented with each update.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param versionAttribute The name of the version attribute
     * @return Map containing the operation result
     */
    public static Map<String, Object> optimisticLockingExample(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> key, String versionAttribute) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Get the current version of the item
            GetItemRequest getRequest = GetItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .projectionExpression(versionAttribute)
                .build();
```

```
            GetItemResponse getResponse = dynamoDbClient.getItem(getRequest);

            // Check if the item exists
            if (getResponse.item() == null || !
getResponse.item().containsKey(versionAttribute)) {
                // Item doesn't exist or doesn't have a version attribute
                // Initialize with version 1
                UpdateItemRequest initRequest = UpdateItemRequest.builder()
                    .tableName(tableName)
                    .key(key)
                    .updateExpression("SET #verAttr = :newVer, #dataAttr
 = :data")
                    .expressionAttributeNames(Map.of("#verAttr",
 versionAttribute, "#dataAttr", "Data"))
                    .expressionAttributeValues(Map.of(
                        ":newVer", AttributeValue.builder().n("1").build(),
                        ":data", AttributeValue.builder().s("Initial
 data").build()))
                    .returnValues(ReturnValue.UPDATED_NEW)
                    .build();

                UpdateItemResponse initResponse =
 dynamoDbClient.updateItem(initRequest);

                result.put("operation", "initialize");
                result.put("success", true);
                result.put("attributes", initResponse.attributes());

                return result;
            }

            // Get the current version number
            int currentVersion =
                Integer.parseInt(getResponse.item().get(versionAttribute).n());
            int newVersion = currentVersion + 1;

            // Update the item with a condition on the version
            UpdateItemRequest updateRequest = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #verAttr = :newVer, #dataAttr = :newData")
                .conditionExpression("#verAttr = :curVer")
```

```
                .expressionAttributeNames(Map.of("#verAttr", versionAttribute,
"#dataAttr", "Data"))
                .expressionAttributeValues(Map.of(
                    ":curVer",
                        AttributeValue.builder()
                            .n(String.valueOf(currentVersion))
                            .build(),
                    ":newVer",

AttributeValue.builder().n(String.valueOf(newVersion)).build(),
                    ":newData",
                        AttributeValue.builder()
                            .s("Updated data at version " + newVersion)
                            .build()))
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            UpdateItemResponse updateResponse =
dynamoDbClient.updateItem(updateRequest);

            // Record success result
            result.put("operation", "update");
            result.put("success", true);
            result.put("oldVersion", currentVersion);
            result.put("newVersion", newVersion);
            result.put("attributes", updateResponse.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to version mismatch
            result.put("operation", "update");
            result.put("success", false);
            result.put("message", "Version mismatch, another process may have
updated the item");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
            // Record failure due to other errors
            result.put("operation", "update");
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
```

```
    }

    /**
     * Performs a conditional update with multiple conditions.
     *
     * <p>This method demonstrates how to use multiple conditions in a condition
expression.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param conditions Map of attribute names to values for conditions
     * @param updateAttribute The attribute to update
     * @param updateValue The new value to set
     * @return Map containing the operation result and status
     */
    public static Map<String, Object> conditionalUpdateWithMultipleConditions(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        Map<String, AttributeValue> conditions,
        String updateAttribute,
        AttributeValue updateValue) {

        Map<String, Object> result = new HashMap<>();

        try {
            // Build the condition expression and attribute names/values
            StringBuilder conditionExpression = new StringBuilder();
            Map<String, String> expressionAttributeNames = new HashMap<>();
            Map<String, AttributeValue> expressionAttributeValues = new
HashMap<>();

            // Add update attribute
            expressionAttributeNames.put("#updateAttr", updateAttribute);
            expressionAttributeValues.put(":updateVal", updateValue);

            // Add conditions
            int i = 0;
            for (Map.Entry<String, AttributeValue> condition :
conditions.entrySet()) {
                String attrName = condition.getKey();
                AttributeValue attrValue = condition.getValue();
```

```
                String nameKey = "#cond" + i;
                String valueKey = ":val" + i;

                expressionAttributeNames.put(nameKey, attrName);
                expressionAttributeValues.put(valueKey, attrValue);

                // Add AND between conditions (except for the first one)
                if (i > 0) {
                    conditionExpression.append(" AND ");
                }

                conditionExpression.append(nameKey).append(" =
").append(valueKey);
                i++;
            }

            // Define the update parameters
            UpdateItemRequest request = UpdateItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .updateExpression("SET #updateAttr = :updateVal")
                .conditionExpression(conditionExpression.toString())
                .expressionAttributeNames(expressionAttributeNames)
                .expressionAttributeValues(expressionAttributeValues)
                .returnValues(ReturnValue.UPDATED_NEW)
                .build();

            // Perform the update operation
            UpdateItemResponse response = dynamoDbClient.updateItem(request);

            // Record success result
            result.put("success", true);
            result.put("message", "All conditions were met and update was
performed");
            result.put("attributes", response.attributes());

        } catch (ConditionalCheckFailedException e) {
            // Record failure due to condition not being met
            result.put("success", false);
            result.put("message", "One or more conditions were not met, update
was not performed");
            result.put("error", "ConditionalCheckFailedException");

        } catch (DynamoDbException e) {
```

```
            // Record failure due to other errors
            result.put("success", false);
            result.put("message", "Error occurred: " + e.getMessage());
            result.put("error", e.getClass().getSimpleName());
        }

        return result;
    }
```

Ejemplo de uso de operaciones condicionales con AWS SDK for Java 2.x.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating conditional operations in DynamoDB");

        try {
            // Example 1: Conditional update
            System.out.println("\nExample 1: Conditional update");
            Map<String, Object> updateResult = conditionalUpdate(
                dynamoDbClient,
                tableName,
                key,
                "InStock",
                AttributeValue.builder().bool(true).build(),
                "Status",
                AttributeValue.builder().s("Available").build());

            System.out.println("Update result: " + updateResult.get("message"));
            if ((boolean) updateResult.get("success")) {
                System.out.println("Updated attributes: " +
updateResult.get("attributes"));
            }

            // Example 2: Conditional delete
            System.out.println("\nExample 2: Conditional delete");
            Map<String, Object> deleteResult = conditionalDelete(
                dynamoDbClient,
                tableName,
```

```
                key,
                "Status",
                AttributeValue.builder().s("Discontinued").build());

            System.out.println("Delete result: " + deleteResult.get("message"));
            if ((boolean) deleteResult.get("success")) {
                System.out.println("Deleted item: " +
deleteResult.get("attributes"));
            }

            // Example 3: Optimistic locking
            System.out.println("\nExample 3: Optimistic locking");
            Map<String, Object> lockingResult =
optimisticLockingExample(dynamoDbClient, tableName, key, "Version");

            System.out.println("Optimistic locking result:");
            System.out.println("  Operation: " + lockingResult.get("operation"));
            System.out.println("  Success: " + lockingResult.get("success"));
            if (lockingResult.get("operation").equals("update") && (boolean)
lockingResult.get("success")) {
                System.out.println("  Old version: " +
lockingResult.get("oldVersion"));
                System.out.println("  New version: " +
lockingResult.get("newVersion"));
            }
            System.out.println("  Attributes: " +
lockingResult.get("attributes"));

            // Example 4: Multiple conditions
            System.out.println("\nExample 4: Multiple conditions");
            Map<String, AttributeValue> conditions = new HashMap<>();
            conditions.put("Price",
AttributeValue.builder().n("199.99").build());
            conditions.put("Category",
AttributeValue.builder().s("Electronics").build());

            Map<String, Object> multiConditionResult =
conditionalUpdateWithMultipleConditions(
                dynamoDbClient,
                tableName,
                key,
                conditions,
                "OnSale",
                AttributeValue.builder().bool(true).build());
```

```
            System.out.println("Multiple conditions result: " +
multiConditionResult.get("message"));
            if ((boolean) multiConditionResult.get("success")) {
                System.out.println("Updated attributes: " +
multiConditionResult.get("attributes"));
            }

            // Explain conditional operations
            System.out.println("\nKey points about DynamoDB conditional
operations:");
            System.out.println("1. Conditional operations only succeed if the
condition is met");
            System.out.println("2. ConditionalCheckFailedException is thrown when
the condition fails");
            System.out.println("3. No changes are made to the item if the
condition fails");
            System.out.println("4. Conditions can be used with update, delete,
and put operations");
            System.out.println("5. Multiple conditions can be combined with AND
and OR");
            System.out.println("6. Optimistic locking can be implemented using a
version attribute");
            System.out.println(
                "7. Conditional operations consume the same amount of write
capacity whether they succeed or fail");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for Java 2.x.
  - DeleteItem
  - PutItem
  - UpdateItem

JavaScript

SDK para JavaScript (v3)

Demuestre las operaciones condicionales con AWS SDK para JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  DeleteCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Perform a conditional update operation.
 *
 * This function demonstrates how to update an item only if a condition is met.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} conditionAttribute - The attribute to check in the condition
 * @param {any} conditionValue - The value to compare against
 * @param {string} updateAttribute - The attribute to update
 * @param {any} updateValue - The new value to set
 * @returns {Promise<Object>} - Result of the operation
 */
async function conditionalUpdate(
  config,
  tableName,
  key,
  conditionAttribute,
  conditionValue,
  updateAttribute,
  updateValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters with a condition expression
  const params = {
```

```
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${updateAttribute} = :value`,
    ConditionExpression: `${conditionAttribute} = :condition`,
    ExpressionAttributeValues: {
      ":value": updateValue,
      ":condition": conditionValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Condition was met and update was performed",
      updatedAttributes: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Condition was not met, update was not performed",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Perform a conditional delete operation.
 *
 * This function demonstrates how to delete an item only if a condition is met.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to delete
 * @param {string} conditionAttribute - The attribute to check in the condition
```

```
 * @param {any} conditionValue - The value to compare against
 * @returns {Promise<Object>} - Result of the operation
 */
async function conditionalDelete(
  config,
  tableName,
  key,
  conditionAttribute,
  conditionValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the delete parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    ConditionExpression: `${conditionAttribute} = :condition`,
    ExpressionAttributeValues: {
      ":condition": conditionValue
    },
    ReturnValues: "ALL_OLD"
  };

  try {
    // Perform the delete operation
    const response = await docClient.send(new DeleteCommand(params));

    return {
      success: true,
      message: "Condition was met and item was deleted",
      deletedItem: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Condition was not met, item was not deleted",
        error: "ConditionalCheckFailedException"
      };
    }
```

```
    // Re-throw other errors
    throw error;
  }
}

/**
 * Implement optimistic locking with a version number.
 *
 * This function demonstrates how to use a version number for optimistic locking
 * to prevent race conditions when multiple processes update the same item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {Object} updates - The attributes to update
 * @param {number} expectedVersion - The expected current version number
 * @returns {Promise<Object>} - Result of the operation
 */
async function updateWithOptimisticLocking(
  config,
  tableName,
  key,
  updates,
  expectedVersion
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression
  const updateExpressions = [];
  const expressionAttributeValues = {
    ":expectedVersion": expectedVersion,
    ":newVersion": expectedVersion + 1
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([attribute, value], index) => {
    updateExpressions.push(`${attribute} = :val${index}`);
    expressionAttributeValues[`:val${index}`] = value;
  });

  // Add the version update
  updateExpressions.push("version = :newVersion");
```

```
  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${updateExpressions.join(", ")}`,
    ConditionExpression: "version = :expectedVersion",
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return {
      success: true,
      message: "Update succeeded with optimistic locking",
      newVersion: expectedVersion + 1,
      updatedAttributes: response.Attributes
    };
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Optimistic locking failed: the item was modified by another
 process",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Implement a conditional write that creates an item only if it doesn't exist.
 *
 * This function demonstrates how to use attribute_not_exists to create an item
 * only if it doesn't already exist (similar to an "INSERT IF NOT EXISTS"
 operation).
 *
```

```
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} item - The item to create
 * @returns {Promise<Object>} - Result of the operation
 */
async function createIfNotExists(
  config,
  tableName,
  item
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Extract the primary key attributes
  const keyAttributes = Object.keys(item).filter(attr =>
    attr === "id" || attr === "ID" || attr === "Id" ||
    attr.endsWith("Id") || attr.endsWith("ID") ||
    attr.endsWith("Key")
  );

  if (keyAttributes.length === 0) {
    throw new Error("Could not determine primary key attributes");
  }

  // Create a condition expression that checks if the item doesn't exist
  const conditionExpression = `attribute_not_exists(${keyAttributes[0]})`;

  // Define the put parameters with a condition expression
  const params = {
    TableName: tableName,
    Item: item,
    ConditionExpression: conditionExpression
  };

  try {
    // Perform the put operation
    await docClient.send(new PutCommand(params));

    return {
      success: true,
      message: "Item was created because it didn't exist",
      item
    };
```

```
  } catch (error) {
    // Check if the error is due to the condition check failing
    if (error.name === "ConditionalCheckFailedException") {
      return {
        success: false,
        message: "Item already exists, creation was skipped",
        error: "ConditionalCheckFailedException"
      };
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
```

```
  return response.Item || null;
}
```

Ejemplo de uso de operaciones condicionales con AWS SDK para JavaScript.

```
/**
 * Example of how to use conditional operations.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating conditional operations in DynamoDB");

  try {
    // Example 1: Conditional update based on attribute value
    console.log("\nExample 1: Conditional update based on attribute value");
    const updateResult = await conditionalUpdate(
      config,
      tableName,
      key,
      "Category",
      "Electronics",
      "Price",
      299.99
    );

    console.log(`Result: ${updateResult.message}`);
    if (updateResult.success) {
      console.log("Updated attributes:", updateResult.updatedAttributes);
    }

    // Example 2: Conditional delete based on attribute value
    console.log("\nExample 2: Conditional delete based on attribute value");
    const deleteResult = await conditionalDelete(
      config,
      tableName,
      key,
      "InStock",
      false
```

```
    );

    console.log(`Result: ${deleteResult.message}`);
    if (deleteResult.success) {
      console.log("Deleted item:", deleteResult.deletedItem);
    }

    // Example 3: Optimistic locking with version number
    console.log("\nExample 3: Optimistic locking with version number");

    // First, get the current item to check its version
    const currentItem = await getItem(config, tableName, { ProductId:
"P67890" });
    const currentVersion = currentItem ? (currentItem.version || 0) : 0;

    console.log(`Current version: ${currentVersion}`);

    // Then, update with optimistic locking
    const lockingResult = await updateWithOptimisticLocking(
      config,
      tableName,
      { ProductId: "P67890" },
      {
        Name: "Updated Product Name",
        Description: "This is an updated description"
      },
      currentVersion
    );

    console.log(`Result: ${lockingResult.message}`);
    if (lockingResult.success) {
      console.log(`New version: ${lockingResult.newVersion}`);
      console.log("Updated attributes:", lockingResult.updatedAttributes);
    }

    // Example 4: Create item only if it doesn't exist
    console.log("\nExample 4: Create item only if it doesn't exist");
    const createResult = await createIfNotExists(
      config,
      tableName,
      {
        ProductId: "P99999",
        Name: "New Product",
        Category: "Accessories",
```

```
        Price: 19.99,
        InStock: true
      }
    );

    console.log(`Result: ${createResult.message}`);
    if (createResult.success) {
      console.log("Created item:", createResult.item);
    }

    // Explain conditional operations
    console.log("\nKey points about conditional operations:");
    console.log("1. Conditional operations only succeed if the condition is
 met");
    console.log("2. ConditionalCheckFailedException indicates the condition
 wasn't met");
    console.log("3. Optimistic locking prevents race conditions in concurrent
 updates");
    console.log("4. attribute_exists and attribute_not_exists are useful for
 checking if attributes are present");
    console.log("5. Conditional operations are atomic - they either succeed
 completely or fail completely");
    console.log("6. You can use any valid comparison operators and functions in
 condition expressions");
    console.log("7. Conditional operations don't consume write capacity if the
 condition fails");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para JavaScript.
  - DeleteItem
  - PutItem
  - UpdateItem

Python

SDK para Python (Boto3)

Demuestre las operaciones condicionales con AWS SDK for Python (Boto3).

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, Optional, Tuple, Union


def conditional_update(
    table_name: str,
    key: Dict[str, Any],
    condition_attribute: str,
    condition_value: Any,
    update_attribute: str,
    update_value: Any,
) -> Tuple[bool, Optional[Dict[str, Any]]]:
    """
    Update an item only if a condition is met.

    This function demonstrates how to perform a conditional update operation
    and determine if the condition was met.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        condition_attribute (str): The attribute to check in the condition.
        condition_value (Any): The value to compare against.
        update_attribute (str): The attribute to update.
        update_value (Any): The new value to set.

    Returns:
        Tuple[bool, Optional[Dict[str, Any]]]: A tuple containing:
            - A boolean indicating if the update succeeded
            - The response from DynamoDB if successful, None otherwise
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        # Perform the conditional update
```

```
        response = table.update_item(
            Key=key,
            UpdateExpression="SET #update_attr = :update_val",
            ConditionExpression="#cond_attr = :cond_val",
            ExpressionAttributeNames={
                "#update_attr": update_attribute,
                "#cond_attr": condition_attribute,
            },
            ExpressionAttributeValues={":update_val": update_value, ":cond_val":
 condition_value},
            ReturnValues="UPDATED_NEW",
        )
        # Update succeeded, condition was met
        return True, response
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return False, None
        else:
            # Other error occurred
            raise


def conditional_delete(
    table_name: str, key: Dict[str, Any], condition_attribute: str,
 condition_value: Any
) -> bool:
    """
    Delete an item only if a condition is met.

    This function demonstrates how to perform a conditional delete operation
    and determine if the condition was met.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to delete.
        condition_attribute (str): The attribute to check in the condition.
        condition_value (Any): The value to compare against.

    Returns:
        bool: True if the delete succeeded (condition was met), False otherwise.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
```

```
    table = dynamodb.Table(table_name)

    try:
        # Perform the conditional delete
        table.delete_item(
            Key=key,
            ConditionExpression="#attr = :val",
            ExpressionAttributeNames={"#attr": condition_attribute},
            ExpressionAttributeValues={":val": condition_value},
        )
        # Delete succeeded, condition was met
        return True
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return False
        else:
            # Other error occurred
            raise


def optimistic_locking_update(
    table_name: str,
    key: Dict[str, Any],
    version_attribute: str,
    update_attribute: str,
    update_value: Any,
) -> Tuple[bool, Optional[Dict[str, Any]]]:
    """
    Update an item using optimistic locking with a version attribute.

    This function demonstrates how to implement optimistic locking using
    a version attribute that is incremented with each update.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        version_attribute (str): The name of the version attribute.
        update_attribute (str): The attribute to update.
        update_value (Any): The new value to set.

    Returns:
        Tuple[bool, Optional[Dict[str, Any]]]: A tuple containing:
            - A boolean indicating if the update succeeded
```

```
            - The response from DynamoDB if successful, None otherwise
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # First, get the current version
    try:
        response = table.get_item(
            Key=key,
            ProjectionExpression=f"#{version_attribute}",
            ExpressionAttributeNames={f"#{version_attribute}":
version_attribute},
        )

        item = response.get("Item", {})
        current_version = item.get(version_attribute, 0)

        # Now, try to update with a condition on the version
        try:
            update_response = table.update_item(
                Key=key,
                UpdateExpression=f"SET #{update_attribute} = :update_val,
#{version_attribute} = :new_version",
                ConditionExpression=f"#{version_attribute} = :current_version",
                ExpressionAttributeNames={
                    f"#{update_attribute}": update_attribute,
                    f"#{version_attribute}": version_attribute,
                },
                ExpressionAttributeValues={
                    ":update_val": update_value,
                    ":current_version": current_version,
                    ":new_version": current_version + 1,
                },
                ReturnValues="UPDATED_NEW",
            )
            # Update succeeded
            return True, update_response
        except ClientError as e:
            if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
                # Version has changed, optimistic locking failed
                return False, None
            else:
                # Other error occurred
```

```
                raise
    except ClientError:
        # Error getting the item
        raise


def conditional_check_and_update(
    table_name: str,
    key: Dict[str, Any],
    check_attribute: str,
    check_value: Any,
    update_attribute: str,
    update_value: Any,
    create_if_not_exists: bool = False,
) -> Union[Dict[str, Any], None]:
    """
    Check if an attribute has a specific value and update another attribute if it
 does.

    This function demonstrates a more complex conditional update that can also
    create the item if it doesn't exist.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        check_attribute (str): The attribute to check in the condition.
        check_value (Any): The value to compare against.
        update_attribute (str): The attribute to update.
        update_value (Any): The new value to set.
        create_if_not_exists (bool, optional): Whether to create the item if it
 doesn't exist.

    Returns:
        Union[Dict[str, Any], None]: The response from DynamoDB if successful,
 None otherwise.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    try:
        if create_if_not_exists:
            # Use attribute_not_exists to create the item if it doesn't exist
```

```
            condition_expression = "attribute_not_exists(#pk) OR #check_attr
= :check_val"
            update_expression = "SET #update_attr = :update_val, #check_attr =
if_not_exists(#check_attr, :check_val)"

            # Get the partition key name from the key dictionary
            pk_name = next(iter(key))

            expression_attribute_names = {
                "#pk": pk_name,
                "#check_attr": check_attribute,
                "#update_attr": update_attribute,
            }
        else:
            # Only update if the check attribute has the expected value
            condition_expression = "#check_attr = :check_val"
            update_expression = "SET #update_attr = :update_val"

            expression_attribute_names = {
                "#check_attr": check_attribute,
                "#update_attr": update_attribute,
            }

        # Perform the conditional update
        response = table.update_item(
            Key=key,
            UpdateExpression=update_expression,
            ConditionExpression=condition_expression,
            ExpressionAttributeNames=expression_attribute_names,
            ExpressionAttributeValues={":check_val": check_value, ":update_val":
update_value},
            ReturnValues="UPDATED_NEW",
        )
        return response
    except ClientError as e:
        if e.response["Error"]["Code"] == "ConditionalCheckFailedException":
            # Condition was not met
            return None
        else:
            # Other error occurred
            raise
```

Ejemplo de uso de operaciones condicionales con AWS SDK for Python (Boto3).

```
def example_usage():
    """Example of how to use the conditional operations functions."""
    # Example parameters
    table_name = "Products"
    key = {"ProductId": "prod123"}

    print("Example 1: Conditional Update")
    try:
        # Update the price only if the current stock is greater than 10
        success, response = conditional_update(
            table_name=table_name,
            key=key,
            condition_attribute="Stock",
            condition_value=10,
            update_attribute="Price",
            update_value=99.99,
        )

        if success:
            # Fix for mypy: Handle the case where response might be None
            attributes = {} if response is None else response.get("Attributes",
 {})
            print(f"Update succeeded! New values: {attributes}")
        else:
            print("Update failed because the condition was not met.")
    except Exception as e:
        print(f"Error during conditional update: {e}")

    print("\nExample 2: Conditional Delete")
    try:
        # Delete the product only if it's discontinued
        success = conditional_delete(
            table_name=table_name,
            key=key,
            condition_attribute="Status",
            condition_value="Discontinued",
        )

        if success:
```

```
            print("Delete succeeded! The item was deleted.")
        else:
            print("Delete failed because the condition was not met.")
    except Exception as e:
        print(f"Error during conditional delete: {e}")

    print("\nExample 3: Optimistic Locking")
    try:
        # Update with optimistic locking using a version attribute
        success, response = optimistic_locking_update(
            table_name=table_name,
            key=key,
            version_attribute="Version",
            update_attribute="Description",
            update_value="Updated product description",
        )

        if success:
            # Fix for mypy: Handle the case where response might be None
            attributes = {} if response is None else response.get("Attributes",
{})
            print(f"Optimistic locking update succeeded! New values:
{attributes}")
        else:
            print("Optimistic locking update failed because the version has
changed.")
    except Exception as e:
        print(f"Error during optimistic locking update: {e}")

    print("\nExample 4: Conditional Check and Update")
    try:
        # Update the featured status if the product is in stock
        response = conditional_check_and_update(
            table_name=table_name,
            key=key,
            check_attribute="InStock",
            check_value=True,
            update_attribute="Featured",
            update_value=True,
            create_if_not_exists=True,
        )

        if response:
            print(
```

```
                f"Conditional check and update succeeded! New values:
{response.get('Attributes', {})}"
            )
        else:
            print("Conditional check and update failed because the condition was
not met.")
    except Exception as e:
        print(f"Error during conditional check and update: {e}")

    print("\nUnderstanding Conditional Operations in DynamoDB:")
    print("1. Conditional operations help maintain data integrity")
    print("2. They prevent race conditions in concurrent environments")
    print("3. Failed conditions result in ConditionalCheckFailedException")
    print("4. No DynamoDB capacity is consumed when conditions fail")
    print("5. Optimistic locking is a common pattern using version attributes")
    print("6. Conditions can be combined with logical operators (AND, OR, NOT)")
    print("7. Conditions can use comparison operators (=, <>, <, <=, >, >=)")
    print(
        "8. attribute_exists() and attribute_not_exists() are useful for checking
attribute presence"
    )
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Python (Boto3).
  - DeleteItem
  - PutItem
  - UpdateItem

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de nombres de atributo de expresión en DynamoDB con un AWS SDK

En los siguientes ejemplos de código se muestra cómo utilizar nombres de atributo de expresión en DynamoDB.

- Trabaje con palabras reservadas en expresiones de DynamoDB.

- Utilice marcadores de posición para los nombres de atributos de expresión.
- Gestione caracteres especiales en nombres de atributo.

Java

SDK para Java 2.x

Demuestre los nombres de atributo de expresión con AWS SDK for Java 2.x.

```
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.QueryRequest;
import software.amazon.awssdk.services.dynamodb.model.QueryResponse;
import software.amazon.awssdk.services.dynamodb.model.ScanRequest;
import software.amazon.awssdk.services.dynamodb.model.ScanResponse;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemRequest;
import software.amazon.awssdk.services.dynamodb.model.UpdateItemResponse;

import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

    /**
     * Updates an attribute that is a reserved word in DynamoDB.
     *
     * <p>This method demonstrates how to use expression attribute names to
 update
     * attributes that are reserved words in DynamoDB.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param reservedWordAttribute The reserved word attribute to update
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateReservedWordAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
```

```
        Map<String, AttributeValue> key,
        String reservedWordAttribute,
        AttributeValue value) {

        // Define the update parameters using expression attribute names
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET #attr = :value")
            .expressionAttributeNames(Map.of("#attr", reservedWordAttribute))
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Updates an attribute that contains special characters.
     *
     * <p>This method demonstrates how to use expression attribute names to
update
     * attributes that contain special characters.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param specialCharAttribute The attribute with special characters to
update
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateSpecialCharacterAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        String specialCharAttribute,
        AttributeValue value) {

        // Define the update parameters using expression attribute names
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
```

```
            .key(key)
            .updateExpression("SET #attr = :value")
            .expressionAttributeNames(Map.of("#attr", specialCharAttribute))
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Queries items using an attribute that is a reserved word.
     *
     * <p>This method demonstrates how to use expression attribute names in a
query
     * when the attribute is a reserved word.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param partitionKeyName The name of the partition key attribute
     * @param partitionKeyValue The value of the partition key
     * @param reservedWordAttribute The reserved word attribute to filter on
     * @param value The value to compare against
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static QueryResponse queryWithReservedWordAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        String partitionKeyName,
        AttributeValue partitionKeyValue,
        String reservedWordAttribute,
        AttributeValue value) {

        // Define the query parameters using expression attribute names
        Map<String, String> expressionAttributeNames = new HashMap<>();
        expressionAttributeNames.put("#pkName", partitionKeyName);
        expressionAttributeNames.put("#attr", reservedWordAttribute);

        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        expressionAttributeValues.put(":pkValue", partitionKeyValue);
        expressionAttributeValues.put(":value", value);
```

```
        QueryRequest request = QueryRequest.builder()
            .tableName(tableName)
            .keyConditionExpression("#pkName = :pkValue")
            .filterExpression("#attr = :value")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the query operation
        return dynamoDbClient.query(request);
    }

    /**
     * Updates a nested attribute with a path that contains reserved words.
     *
     * <p>This method demonstrates how to use expression attribute names to
update
     * nested attributes where the path contains reserved words.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param key The key of the item to update
     * @param attributePath The path to the nested attribute as an array
     * @param value The value to set
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static UpdateItemResponse updateNestedReservedWordAttribute(
        DynamoDbClient dynamoDbClient,
        String tableName,
        Map<String, AttributeValue> key,
        List<String> attributePath,
        AttributeValue value) {

        // Create expression attribute names for each part of the path
        Map<String, String> expressionAttributeNames = new HashMap<>();
        for (int i = 0; i < attributePath.size(); i++) {
            expressionAttributeNames.put("#attr" + i, attributePath.get(i));
        }

        // Build the attribute path using the expression attribute names
        StringBuilder attributePathExpression = new StringBuilder();
        for (int i = 0; i < attributePath.size(); i++) {
            if (i > 0) {
```

```
                attributePathExpression.append(".");
            }
            attributePathExpression.append("#attr").append(i);
        }

        // Define the update parameters
        UpdateItemRequest request = UpdateItemRequest.builder()
            .tableName(tableName)
            .key(key)
            .updateExpression("SET " + attributePathExpression.toString() + "
= :value")
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(Map.of(":value", value))
            .returnValues("UPDATED_NEW")
            .build();

        // Perform the update operation
        return dynamoDbClient.updateItem(request);
    }

    /**
     * Scans a table with multiple attribute name placeholders.
     *
     * <p>This method demonstrates how to use multiple expression attribute names
     * in a complex filter expression.
     *
     * @param dynamoDbClient The DynamoDB client
     * @param tableName The name of the DynamoDB table
     * @param filters Object mapping attribute names to filter values
     * @return The response from DynamoDB
     * @throws DynamoDbException if an error occurs during the operation
     */
    public static ScanResponse scanWithMultipleAttributeNames(
        DynamoDbClient dynamoDbClient, String tableName, Map<String,
AttributeValue> filters) {

        // Create expression attribute names and values
        Map<String, String> expressionAttributeNames = new HashMap<>();
        Map<String, AttributeValue> expressionAttributeValues = new HashMap<>();
        StringBuilder filterExpression = new StringBuilder();

        // Build the filter expression
        int index = 0;
        for (Map.Entry<String, AttributeValue> entry : filters.entrySet()) {
```

```
            String attrName = entry.getKey();
            AttributeValue attrValue = entry.getValue();

            String nameKey = "#attr" + index;
            String valueKey = ":val" + index;

            expressionAttributeNames.put(nameKey, attrName);
            expressionAttributeValues.put(valueKey, attrValue);

            // Add AND between conditions (except for the first one)
            if (index > 0) {
                filterExpression.append(" AND ");
            }

            filterExpression.append(nameKey).append(" = ").append(valueKey);
            index++;
        }

        // Define the scan parameters
        ScanRequest request = ScanRequest.builder()
            .tableName(tableName)
            .filterExpression(filterExpression.toString())
            .expressionAttributeNames(expressionAttributeNames)
            .expressionAttributeValues(expressionAttributeValues)
            .build();

        // Perform the scan operation
        return dynamoDbClient.scan(request);
    }
```

Ejemplo de uso de nombres de atributo de expresión con AWS SDK for Java 2.x.

```
    public static void exampleUsage(DynamoDbClient dynamoDbClient, String
tableName) {
        // Example key
        Map<String, AttributeValue> key = new HashMap<>();
        key.put("ProductId", AttributeValue.builder().s("P12345").build());

        System.out.println("Demonstrating expression attribute names in
DynamoDB");

        try {
```

```
            // Example 1: Update an attribute that is a reserved word
            System.out.println("\nExample 1: Updating an attribute that is a
reserved word");
            UpdateItemResponse response1 = updateReservedWordAttribute(
                dynamoDbClient,
                tableName,
                key,
                "Size", // "SIZE" is a reserved word in DynamoDB
                AttributeValue.builder().s("Large").build());

            System.out.println("Updated attribute: " + response1.attributes());

            // Example 2: Update an attribute with special characters
            System.out.println("\nExample 2: Updating an attribute with special
characters");
            UpdateItemResponse response2 = updateSpecialCharacterAttribute(
                dynamoDbClient,
                tableName,
                key,
                "Product-Type", // Contains a hyphen, which is a special
character
                AttributeValue.builder().s("Electronics").build());

            System.out.println("Updated attribute: " + response2.attributes());

            // Example 3: Query with a reserved word attribute
            System.out.println("\nExample 3: Querying with a reserved word
attribute");
            QueryResponse response3 = queryWithReservedWordAttribute(
                dynamoDbClient,
                tableName,
                "Category",
                AttributeValue.builder().s("Electronics").build(),
                "Count", // "COUNT" is a reserved word in DynamoDB
                AttributeValue.builder().n("10").build());

            System.out.println("Found " + response3.count() + " items");

            // Example 4: Update a nested attribute with reserved words in the
path
            System.out.println("\nExample 4: Updating a nested attribute with
reserved words in the path");
            UpdateItemResponse response4 = updateNestedReservedWordAttribute(
                dynamoDbClient,
```

```
                tableName,
                key,
                Arrays.asList("Dimensions", "Size", "Height"), // "SIZE" is a
 reserved word
                AttributeValue.builder().n("30").build());

            System.out.println("Updated nested attribute: " +
 response4.attributes());

            // Example 5: Scan with multiple attribute name placeholders
            System.out.println("\nExample 5: Scanning with multiple attribute
 name placeholders");
            Map<String, AttributeValue> filters = new HashMap<>();
            filters.put("Size", AttributeValue.builder().s("Large").build());
            filters.put("Count", AttributeValue.builder().n("10").build());
            filters.put(
                "Product-Type",
 AttributeValue.builder().s("Electronics").build());

            ScanResponse response5 =
 scanWithMultipleAttributeNames(dynamoDbClient, tableName, filters);

            System.out.println("Found " + response5.count() + " items");

            // Show some common reserved words
            System.out.println("\nSome common DynamoDB reserved words:");
            List<String> commonReservedWords = getDynamoDBReservedWords();
            System.out.println(String.join(", ", commonReservedWords));

            // Explain expression attribute names
            System.out.println("\nKey points about expression attribute names:");
            System.out.println("1. Use expression attribute names (#name) for
 reserved words");
            System.out.println("2. Use expression attribute names for attributes
 with special characters");
            System.out.println(
                "3. Special characters include: spaces, hyphens, dots, and other
 non-alphanumeric characters");
            System.out.println("4. Expression attribute names are required for
 nested attributes with reserved words");
            System.out.println("5. You can use multiple expression attribute
 names in a single expression");
            System.out.println("6. Expression attribute names are case-
sensitive");
```

```
            System.out.println("7. Expression attribute names are only used in
 expressions, not in the actual data");

        } catch (DynamoDbException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK for Java 2.x.
  - Query
  - UpdateItem

JavaScript

SDK para JavaScript (v3)

Demuestre los nombres de atributo de expresión con AWS SDK para JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  QueryCommand,
  ScanCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an attribute that is a reserved word in DynamoDB.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that are reserved words in DynamoDB.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} reservedWordAttribute - The reserved word attribute to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
```

```
 */
async function updateReservedWordAttribute(
  config,
  tableName,
  key,
  reservedWordAttribute,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using expression attribute names
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: "SET #attr = :value",
    ExpressionAttributeNames: {
      "#attr": reservedWordAttribute
    },
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update an attribute that contains special characters.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that contain special characters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} specialCharAttribute - The attribute with special characters
 to update
 * @param {any} value - The value to set
```

```
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateSpecialCharacterAttribute(
  config,
  tableName,
  key,
  specialCharAttribute,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using expression attribute names
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: "SET #attr = :value",
    ExpressionAttributeNames: {
      "#attr": specialCharAttribute
    },
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Query items using an attribute that is a reserved word.
 *
 * This function demonstrates how to use expression attribute names in a query
 * when the attribute is a reserved word.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key attribute
 * @param {any} partitionKeyValue - The value of the partition key
```

```
 * @param {string} reservedWordAttribute - The reserved word attribute to filter
 on
 * @param {any} value - The value to compare against
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function queryWithReservedWordAttribute(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  reservedWordAttribute,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the query parameters using expression attribute names
  const params = {
    TableName: tableName,
    KeyConditionExpression: "#pkName = :pkValue",
    FilterExpression: "#attr = :value",
    ExpressionAttributeNames: {
      "#pkName": partitionKeyName,
      "#attr": reservedWordAttribute
    },
    ExpressionAttributeValues: {
      ":pkValue": partitionKeyValue,
      ":value": value
    }
  };

  // Perform the query operation
  const response = await docClient.send(new QueryCommand(params));

  return response;
}

/**
 * Update a nested attribute with a path that contains reserved words.
 *
 * This function demonstrates how to use expression attribute names to update
 * nested attributes where the path contains reserved words.
 *
```

```
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string[]} attributePath - The path to the nested attribute as an array
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateNestedReservedWordAttribute(
  config,
  tableName,
  key,
  attributePath,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create expression attribute names for each part of the path
  const expressionAttributeNames = {};
  for (let i = 0; i < attributePath.length; i++) {
    expressionAttributeNames[`#attr${i}`] = attributePath[i];
  }

  // Build the attribute path using the expression attribute names
  const attributePathExpression = attributePath
    .map((_, i) => `#attr${i}`)
    .join(".");

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${attributePathExpression} = :value`,
    ExpressionAttributeNames: expressionAttributeNames,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));
```

```
  return response;
}

/**
 * Scan a table with multiple attribute name placeholders.
 *
 * This function demonstrates how to use multiple expression attribute names
 * in a complex filter expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} filters - Object mapping attribute names to filter values
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function scanWithMultipleAttributeNames(
  config,
  tableName,
  filters
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create expression attribute names and values
  const expressionAttributeNames = {};
  const expressionAttributeValues = {};
  const filterConditions = [];

  // Build the filter expression
  Object.entries(filters).forEach(([attrName, value], index) => {
    const nameKey = `#attr${index}`;
    const valueKey = `:val${index}`;

    expressionAttributeNames[nameKey] = attrName;
    expressionAttributeValues[valueKey] = value;
    filterConditions.push(`${nameKey} = ${valueKey}`);
  });

  // Join the filter conditions with AND
  const filterExpression = filterConditions.join(" AND ");

  // Define the scan parameters
  const params = {
    TableName: tableName,
```

```
    FilterExpression: filterExpression,
    ExpressionAttributeNames: expressionAttributeNames,
    ExpressionAttributeValues: expressionAttributeValues
  };

  // Perform the scan operation
  const response = await docClient.send(new ScanCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Ejemplo de uso de nombres de atributo de expresión con AWS SDK para JavaScript.

```
/**
 * Example of how to use expression attribute names.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating expression attribute names in DynamoDB");

  try {
    // Example 1: Update an attribute that is a reserved word
    console.log("\nExample 1: Updating an attribute that is a reserved word");
    const response1 = await updateReservedWordAttribute(
      config,
      tableName,
      key,
      "Size", // "SIZE" is a reserved word in DynamoDB
      "Large"
    );

    console.log("Updated attribute:", response1.Attributes);

    // Example 2: Update an attribute with special characters
    console.log("\nExample 2: Updating an attribute with special characters");
    const response2 = await updateSpecialCharacterAttribute(
      config,
      tableName,
      key,
      "Product-Type", // Contains a hyphen, which is a special character
      "Electronics"
    );

    console.log("Updated attribute:", response2.Attributes);

    // Example 3: Query with a reserved word attribute
    console.log("\nExample 3: Querying with a reserved word attribute");
    const response3 = await queryWithReservedWordAttribute(
      config,
      tableName,
      "Category",
```

```
      "Electronics",
      "Count", // "COUNT" is a reserved word in DynamoDB
      10
    );

    console.log(`Found ${response3.Items.length} items`);

    // Example 4: Update a nested attribute with reserved words in the path
    console.log("\nExample 4: Updating a nested attribute with reserved words in
the path");
    const response4 = await updateNestedReservedWordAttribute(
      config,
      tableName,
      key,
      ["Dimensions", "Size", "Height"], // "SIZE" is a reserved word
      30
    );

    console.log("Updated nested attribute:", response4.Attributes);

    // Example 5: Scan with multiple attribute name placeholders
    console.log("\nExample 5: Scanning with multiple attribute name
placeholders");
    const response5 = await scanWithMultipleAttributeNames(
      config,
      tableName,
      {
        "Size": "Large",
        "Count": 10,
        "Product-Type": "Electronics"
      }
    );

    console.log(`Found ${response5.Items.length} items`);

    // Get the final state of the item
    console.log("\nFinal state of the item:");
    const item = await getItem(config, tableName, key);
    console.log(JSON.stringify(item, null, 2));

    // Show some common reserved words
    console.log("\nSome common DynamoDB reserved words:");
    const commonReservedWords = [
      "ABORT", "ABSOLUTE", "ACTION", "ADD", "ALL", "ALTER", "AND", "ANY", "AS",
```

```
      "ASC", "BETWEEN", "BY", "CASE", "CAST", "COLUMN", "CONNECT", "COUNT",
      "CREATE", "CURRENT", "DATE", "DELETE", "DESC", "DROP", "ELSE", "EXISTS",
      "FOR", "FROM", "GRANT", "GROUP", "HAVING", "IN", "INDEX", "INSERT", "INTO",
      "IS", "JOIN", "KEY", "LEVEL", "LIKE", "LIMIT", "LOCAL", "MAX", "MIN",
 "NAME",
      "NOT", "NULL", "OF", "ON", "OR", "ORDER", "OUTER", "REPLACE", "RETURN",
      "SELECT", "SET", "SIZE", "TABLE", "THEN", "TO", "UPDATE", "USER", "VALUES",
      "VIEW", "WHERE"
    ];
    console.log(commonReservedWords.join(", "));

    // Explain expression attribute names
    console.log("\nKey points about expression attribute names:");
    console.log("1. Use expression attribute names (#name) for reserved words");
    console.log("2. Use expression attribute names for attributes with special
 characters");
    console.log("3. Special characters include: spaces, hyphens, dots, and other
 non-alphanumeric characters");
    console.log("4. Expression attribute names are required for nested attributes
 with reserved words");
    console.log("5. You can use multiple expression attribute names in a single
 expression");
    console.log("6. Expression attribute names are case-sensitive");
    console.log("7. Expression attribute names are only used in expressions, not
 in the actual data");

  } catch (error) {
    console.error("Error:", error);
  }
}
```

- Para obtener detalles sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para JavaScript.
  - Query
  - UpdateItem

Python

SDK para Python (Boto3)

Demuestre los nombres de atributo de expresión con AWS SDK for Python (Boto3).

```
import boto3
from botocore.exceptions import ClientError
from typing import Any, Dict, List


def use_reserved_word_attribute(
    table_name: str, key: Dict[str, Any], reserved_word: str, value: Any
) -> Dict[str, Any]:
    """
    Update an attribute whose name is a DynamoDB reserved word.

    This function demonstrates how to use expression attribute names to work with
    attributes that have names that are DynamoDB reserved words.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        reserved_word (str): The reserved word to use as an attribute name.
        value (Any): The value to set for the attribute.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use expression attribute names to handle the reserved word
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #reserved_attr = :value",
        ExpressionAttributeNames={"#reserved_attr": reserved_word},
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def use_special_character_attribute(
    table_name: str, key: Dict[str, Any], attribute_with_special_chars: str,
 value: Any
```

```
) -> Dict[str, Any]:
    """
    Update an attribute whose name contains special characters.

    This function demonstrates how to use expression attribute names to work with
    attributes that have names containing special characters like spaces, dots,
 or hyphens.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        attribute_with_special_chars (str): The attribute name with special
 characters.
        value (Any): The value to set for the attribute.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
 attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use expression attribute names to handle special characters
    response = table.update_item(
        Key=key,
        UpdateExpression="SET #special_attr = :value",
        ExpressionAttributeNames={"#special_attr": attribute_with_special_chars},
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )

    return response


def query_with_attribute_names(
    table_name: str,
    partition_key_name: str,
    partition_key_value: str,
    filter_attribute_name: str,
    filter_value: Any,
) -> Dict[str, Any]:
    """
```

```
    Query a table using expression attribute names for both key and filter
 attributes.

    This function demonstrates how to use expression attribute names in a query
 operation
    for both the key condition expression and filter expression.

    Args:
        table_name (str): The name of the DynamoDB table.
        partition_key_name (str): The name of the partition key attribute.
        partition_key_value (str): The value of the partition key to query.
        filter_attribute_name (str): The name of the attribute to filter on.
        filter_value (Any): The value to compare against in the filter.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the query results.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Use expression attribute names for both key condition and filter
    response = table.query(
        KeyConditionExpression="#pk = :pk_val",
        FilterExpression="#filter_attr = :filter_val",
        ExpressionAttributeNames={"#pk": partition_key_name, "#filter_attr":
 filter_attribute_name},
        ExpressionAttributeValues={":pk_val": partition_key_value, ":filter_val":
 filter_value},
    )

    return response


def update_nested_attribute_with_dots(
    table_name: str, key: Dict[str, Any], path_with_dots: str, value: Any
) -> Dict[str, Any]:
    """
    Update a nested attribute using a path with dot notation.

    This function demonstrates how to use expression attribute names to work with
    nested attributes specified using dot notation.

    Args:
```

```
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.
        path_with_dots (str): The path to the nested attribute using dot notation
(e.g., "a.b.c").
        value (Any): The value to set for the nested attribute.

    Returns:
        Dict[str, Any]: The response from DynamoDB containing the updated
attribute values.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Split the path into components
    path_parts = path_with_dots.split(".")

    # Build the update expression and attribute names
    update_expression = "SET "
    expression_attribute_names = {}

    # Build the path expression
    path_expression = ""
    for i, part in enumerate(path_parts):
        name_placeholder = f"#attr{i}"
        expression_attribute_names[name_placeholder] = part

        if i == 0:
            path_expression = name_placeholder
        else:
            path_expression += f".{name_placeholder}"

    # Complete the update expression
    update_expression += f"{path_expression} = :value"

    # Execute the update
    response = table.update_item(
        Key=key,
        UpdateExpression=update_expression,
        ExpressionAttributeNames=expression_attribute_names,
        ExpressionAttributeValues={":value": value},
        ReturnValues="UPDATED_NEW",
    )
```

```
    return response


def demonstrate_attribute_name_requirements(table_name: str, key: Dict[str, Any])
 -> Dict[str, Any]:
    """
    Demonstrate the requirements and allowed characters for attribute names.

    This function shows examples of valid and invalid attribute names and how to
    handle them using expression attribute names.

    Args:
        table_name (str): The name of the DynamoDB table.
        key (Dict[str, Any]): The primary key of the item to update.

    Returns:
        Dict[str, Any]: A dictionary containing the results of the demonstration.
    """
    # Initialize the DynamoDB resource
    dynamodb = boto3.resource("dynamodb")
    table = dynamodb.Table(table_name)

    # Examples of attribute names with different characteristics
    examples = {
        "valid_standard": "NormalAttribute",  # Standard attribute name (no
 placeholder needed)
        "valid_with_underscore": "Normal_Attribute",  # Underscore is allowed
        "valid_with_number": "Attribute123",  # Numbers are allowed
        "reserved_word": "Timestamp",  # Reserved word (requires placeholder)
        "starts_with_number": "123Attribute",  # Starts with number (valid but
 may need placeholder in some contexts)
        "with_space": "Attribute Name",  # Contains space (requires placeholder)
        "with_dot": "Attribute.Name",  # Contains dot (requires placeholder)
        "with_hyphen": "Attribute-Name",  # Contains hyphen (requires
 placeholder)
        "with_special_chars": "Attribute#$%",  # Contains special characters
 (requires placeholder)
    }

    results = {}

    # Try to update each attribute type
    for example_type, attr_name in examples.items():
        try:
```

```
            # For attributes that don't need placeholders, try direct reference
            if example_type in ["valid_standard", "valid_with_underscore",
"valid_with_number"]:
                try:
                    # Try without expression attribute names first
                    response = table.update_item(
                        Key=key,
                        UpdateExpression=f"SET {attr_name} = :value",
                        ExpressionAttributeValues={":value": f"Value for
{attr_name}"},
                        ReturnValues="UPDATED_NEW",
                    )
                    results[example_type] = {
                        "attribute_name": attr_name,
                        "success": True,
                        "needed_placeholder": False,
                        "response": response,
                    }
                except ClientError:
                    # If direct reference fails, try with placeholder
                    response = table.update_item(
                        Key=key,
                        UpdateExpression="SET #attr = :value",
                        ExpressionAttributeNames={"#attr": attr_name},
                        ExpressionAttributeValues={":value": f"Value for
{attr_name}"},
                        ReturnValues="UPDATED_NEW",
                    )
                    results[example_type] = {
                        "attribute_name": attr_name,
                        "success": True,
                        "needed_placeholder": True,
                        "response": response,
                    }
            else:
                # For attributes that definitely need placeholders
                response = table.update_item(
                    Key=key,
                    UpdateExpression="SET #attr = :value",
                    ExpressionAttributeNames={"#attr": attr_name},
                    ExpressionAttributeValues={":value": f"Value for
{attr_name}"},
                    ReturnValues="UPDATED_NEW",
                )
```

```
                results[example_type] = {
                    "attribute_name": attr_name,
                    "success": True,
                    "needed_placeholder": True,
                    "response": response,
                }
        except ClientError as e:
            results[example_type] = {"attribute_name": attr_name, "success":
 False, "error": str(e)}

    return results
```

Ejemplo de uso de nombres de atributo de expresión con AWS SDK for Python (Boto3).

```
def example_usage():
    """Example of how to use expression attribute names in DynamoDB."""
    # Example parameters
    table_name = "Products"
    key = {"ProductId": "prod123"}

    print("Example 1: Using a reserved word as an attribute name")
    try:
        response = use_reserved_word_attribute(
            table_name=table_name, key=key, reserved_word="Timestamp",
 value="2025-05-14T12:00:00Z"
        )
        print(f"Reserved word attribute updated successfully:
 {response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error updating reserved word attribute: {e}")

    print("\nExample 2: Using an attribute name with special characters")
    try:
        response = use_special_character_attribute(
            table_name=table_name,
            key=key,
            attribute_with_special_chars="Product Info",
            value="Special product information",
        )
```

```
        print(f"Special character attribute updated successfully:
{response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error updating special character attribute: {e}")

    print("\nExample 3: Querying with expression attribute names")
    try:
        response = query_with_attribute_names(
            table_name=table_name,
            partition_key_name="Category",
            partition_key_value="Electronics",
            filter_attribute_name="Price",
            filter_value=500,
        )
        print(
            f"Query with expression attribute names returned
{len(response.get('Items', []))} items"
        )
    except Exception as e:
        print(f"Error querying with expression attribute names: {e}")

    print("\nExample 4: Updating a nested attribute with dot notation")
    try:
        response = update_nested_attribute_with_dots(
            table_name=table_name,
            key=key,
            path_with_dots="Product.Details.Specifications",
            value={"Weight": "2.5 kg", "Dimensions": "30x20x10 cm"},
        )
        print(f"Nested attribute updated successfully:
{response.get('Attributes', {})}")
    except Exception as e:
        print(f"Error updating nested attribute: {e}")

    print("\nExample 5: Demonstrating attribute name requirements")
    try:
        results = demonstrate_attribute_name_requirements(table_name=table_name,
key=key)

        print("Attribute Name Requirements Results:")
        for example_type, result in results.items():
            if result.get("success", False):
                needed_placeholder = result.get("needed_placeholder", True)
                print(
```

```
                    f"  - {example_type}: '{result['attribute_name']}' -
{'Requires' if needed_placeholder else 'Does not require'} placeholder"
                )
            else:
                print(
                    f"  - {example_type}: '{result['attribute_name']}' - Failed:
{result.get('error', 'Unknown error')}"
                )
    except Exception as e:
        print(f"Error demonstrating attribute name requirements: {e}")

    print("\nCommon DynamoDB Reserved Words (sample):")
    reserved_words = get_common_reserved_words()
    print(", ".join(reserved_words[:20]) + "... (and many more)")

    print("\nWhen to Use Expression Attribute Names:")
    print("1. When the attribute name is a DynamoDB reserved word")
    print("2. When the attribute name contains special characters (spaces, dots,
hyphens)")
    print("3. When the attribute name begins with a number")
    print("4. When working with nested attributes using dot notation")
    print("5. When you need to reference the same attribute multiple times in an
expression")

    print("\nExpression Attribute Name Requirements:")
    print("1. Must begin with a pound sign (#)")
    print("2. After the pound sign, must contain at least one character")
    print("3. Can contain alphanumeric characters and underscore (_)")
    print("4. Are case-sensitive")
    print("5. Must be unique within a single expression")

    print("\nAttribute Name Requirements in DynamoDB:")
    print("1. Can begin with a-z, A-Z, or 0-9")
    print("2. Can contain a-z, A-Z, 0-9, underscore (_), dash (-), and dot (.)")
    print("3. Are case-sensitive")
    print("4. No length restrictions, but practical limits apply")
    print("5. Cannot be a DynamoDB reserved word if used directly in
expressions")
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de la API de AWS SDK para Python (Boto3).
  - Query
  - UpdateItem

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Uso de eventos programados para invocar una función de Lambda

Los siguientes ejemplos de código muestran cómo crear una función de AWS Lambda invocada por un evento programado de Amazon EventBridge.

Java

SDK para Java 2.x

Muestra cómo crear un evento programado de Amazon EventBridge que invoque una función de AWS Lambda. Configuración de EventBridge para que utilice una expresión cron para programar la invocación de la función de Lambda En este ejemplo, creará una función de Lambda utilizando la API de tiempo de ejecución de Java de Lambda. Este ejemplo invoca diferentes servicios de AWS para realizar un caso de uso específico. Este ejemplo indica cómo crear una aplicación que envíe un mensaje de texto a sus empleados para felicitarles por su primer aniversario.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Registros de CloudWatch
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDK para JavaScript (v3)

Muestra cómo crear un evento programado de Amazon EventBridge que invoque una función de Lambda. Configuración de EventBridge para que utilice una expresión cron para programar la invocación de la función de Lambda En este ejemplo, creará una función de Lambda utilizando la API de tiempo de ejecución de JavaScript de Lambda. Este ejemplo invoca diferentes servicios de AWS para realizar un caso de uso específico. Este ejemplo indica cómo crear una aplicación que envíe un mensaje de texto a sus empleados para felicitarles por su primer aniversario.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Este ejemplo también está disponible en la Guía para desarrolladores de AWS SDK para JavaScript v3.

Servicios utilizados en este ejemplo

- Registros de CloudWatch
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Python

SDK para Python (Boto3)

En este ejemplo, se muestra cómo registrar una función de AWS Lambda como destino de un evento de Amazon EventBridge programado. El controlador de Lambda escribe un mensaje sencillo y los datos de eventos completos en Registros de Amazon CloudWatch para recuperarlos posteriormente.

- Implementa una función de Lambda.
- Crea un evento programado de EventBridge y convierte la función de Lambda en el destino.
- Otorga permiso para permitir que EventBridge invoque la función de Lambda.

- Imprime los datos más recientes de Registros de CloudWatch para mostrar el resultado de las invocaciones programadas.
- Limpia todos los recursos creados durante la demostración.

Este ejemplo se puede ver mejor en GitHub. Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- Registros de CloudWatch
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Trabajo con índices secundarios locales de DynamoDB mediante AWS Command Line Interface v2

En el siguiente ejemplo se muestra cómo crear y consultar tablas con índices secundarios locales.

- Cree una tabla con un índice secundario local (LSI).
- Cree una tabla con varios LSI con diferentes tipos de proyección.
- Consulte datos mediante LSI.

Bash

AWS CLI con Bash script

Cree una tabla con un índice secundario local.

```
# Create a table with a Local Secondary Index
aws dynamodb create-table \
    --table-name CustomerOrders \
```

```
    --attribute-definitions \
        AttributeName=CustomerID,AttributeType=S \
        AttributeName=OrderID,AttributeType=S \
        AttributeName=OrderDate,AttributeType=S \
    --key-schema \
        AttributeName=CustomerID,KeyType=HASH \
        AttributeName=OrderID,KeyType=RANGE \
    --local-secondary-indexes \
        "IndexName=OrderDateIndex,\
        KeySchema=[{AttributeName=CustomerID,KeyType=HASH},
{AttributeName=OrderDate,KeyType=RANGE}],\
        Projection={ProjectionType=ALL}" \
    --billing-mode PAY_PER_REQUEST
```

Cree una tabla con varios LSI.

```
# Create a table with multiple Local Secondary Indexes
aws dynamodb create-table \
    --table-name CustomerDetails \
    --attribute-definitions \
        AttributeName=CustomerID,AttributeType=S \
        AttributeName=Name,AttributeType=S \
        AttributeName=Email,AttributeType=S \
        AttributeName=RegistrationDate,AttributeType=S \
    --key-schema \
        AttributeName=CustomerID,KeyType=HASH \
        AttributeName=Name,KeyType=RANGE \
    --local-secondary-indexes \
        "[
            {
                \"IndexName\": \"EmailIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"CustomerID\",\"KeyType\":\"HASH\"},
                    {\"AttributeName\":\"Email\",\"KeyType\":\"RANGE\"}
                ],
                \"Projection\": {\"ProjectionType\":\"INCLUDE\",
\"NonKeyAttributes\":[\"Address\",\"Phone\"]}
            },
            {
                \"IndexName\": \"RegistrationIndex\",
                \"KeySchema\": [
                    {\"AttributeName\":\"CustomerID\",\"KeyType\":\"HASH\"},
```

```
                    {\"AttributeName\":\"RegistrationDate\",\"KeyType\":\"RANGE
\"}
                ],
                \"Projection\": {\"ProjectionType\":\"KEYS_ONLY\"}
            }
        ]" \
    --billing-mode PAY_PER_REQUEST
```

Consulte datos mediante LSI.

```
# Query the OrderDateIndex LSI
aws dynamodb query \
    --table-name CustomerOrders \
    --index-name OrderDateIndex \
    --key-condition-expression "CustomerID = :custId AND OrderDate BETWEEN :date1
 AND :date2" \
    --expression-attribute-values '{
        ":custId": {"S": "C1"},
        ":date1": {"S": "2023-01-01"},
        ":date2": {"S": "2023-02-01"}
    }'

# Query with a filter expression
aws dynamodb query \
    --table-name CustomerOrders \
    --index-name OrderDateIndex \
    --key-condition-expression "CustomerID = :custId" \
    --filter-expression "Amount > :amount" \
    --expression-attribute-values '{
        ":custId": {"S": "C1"},
        ":amount": {"N": "150"}
    }'
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de comandos de AWS CLI.
  - CreateTable
  - Query

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Trabajo con DynamoDB Streams y tiempo de vida mediante AWS Command Line Interface v2

El siguiente ejemplo de código muestra cómo administrar las características de DynamoDB Streams y tiempo de vida.

- Cree una tabla con Streams habilitado.
- Describa Streams.
- Cree una función de Lambda para procesar Streams.
- Habilite TTL en una tabla.
- Agregue elementos con atributos TTL.
- Describa la configuración de TTL.

Bash

AWS CLI con Bash script

Cree una tabla con Streams habilitado.

```
# Create a table with DynamoDB Streams enabled
aws dynamodb create-table \
    --table-name StreamsDemo \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES
```

Describa Streams.

```
# Get information about the stream
aws dynamodb describe-table \
```

```
    --table-name StreamsDemo \
    --query "Table.StreamSpecification"

# Get the stream ARN
STREAM_ARN=$(aws dynamodb describe-table \
    --table-name StreamsDemo \
    --query "Table.LatestStreamArn" \
    --output text)

echo "Stream ARN: $STREAM_ARN"

# Describe the stream
aws dynamodbstreams describe-stream \
    --stream-arn $STREAM_ARN
```

Cree una función de Lambda para Streams.

```
# Step 1: Create an IAM role for the Lambda function
cat > trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

aws iam create-role \
    --role-name DynamoDBStreamsLambdaRole \
    --assume-role-policy-document file://trust-policy.json

# Step 2: Attach permissions to the role
aws iam attach-role-policy \
    --role-name DynamoDBStreamsLambdaRole \
    --policy-arn arn:aws:iam::aws:policy/service-role/
AWSLambdaDynamoDBExecutionRole
```

```
# Step 3: Create a Lambda function (code would be in a separate file)
echo "Lambda function creation would be done separately with appropriate code"

# Step 4: Create an event source mapping
echo "Example command to create event source mapping:"
echo "aws lambda create-event-source-mapping \\"
echo "    --function-name ProcessDynamoDBRecords \\"
echo "    --event-source $STREAM_ARN \\"
echo "    --batch-size 100 \\"
echo "    --starting-position LATEST"
```

Habilite TTL en una tabla.

```
# Create a table for TTL demonstration
aws dynamodb create-table \
    --table-name TTLDemo \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST

# Wait for table to become active
aws dynamodb wait table-exists --table-name TTLDemo

# Enable TTL on the table
aws dynamodb update-time-to-live \
    --table-name TTLDemo \
    --time-to-live-specification "Enabled=true, AttributeName=ExpirationTime"
```

Agregue elementos con atributos TTL.

```
# Calculate expiration time (current time + 1 day in seconds)
EXPIRATION_TIME=$(date -d "+1 day" +%s)

# Add an item with TTL attribute
aws dynamodb put-item \
    --table-name TTLDemo \
    --item '{
        "ID": {"S": "item1"},
```

```
        "Data": {"S": "This item will expire in 1 day"},
        "ExpirationTime": {"N": "'$EXPIRATION_TIME'"}
    }'

# Add an item that expires in 1 hour
EXPIRATION_TIME_HOUR=$(date -d "+1 hour" +%s)
aws dynamodb put-item \
    --table-name TTLDemo \
    --item '{
        "ID": {"S": "item2"},
        "Data": {"S": "This item will expire in 1 hour"},
        "ExpirationTime": {"N": "'$EXPIRATION_TIME_HOUR'"}
    }'
```

Describa la configuración de TTL.

```
# Describe TTL settings for a table
aws dynamodb describe-time-to-live \
    --table-name TTLDemo
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de comandos de AWS CLI.
  - AttachRolePolicy
  - CreateRole
  - CreateTable
  - DescribeTable
  - DescribeTimeToLive
  - PutItem
  - UpdateTimeToLive

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Trabajo con tablas globales de DynamoDB y replicación de varias regiones con coherencia final (MREC) mediante AWS Command Line Interface v2

En el siguiente ejemplo de código, se muestra cómo administrar las tablas globales de DynamoDB con replicación de varias regiones con coherencia final (MREC).

- Cree una tabla con replicación de varias regiones (MREC).
- Coloque y obtenga elementos de las tablas de réplica.
- Elimine réplicas una por una.
- Efectúe una limpieza eliminando la tabla.

Bash

AWS CLI con Bash script

Cree una tabla con replicación de varias regiones.

```
# Step 1: Create a new table (MusicTable) in US East (Ohio), with DynamoDB
 Streams enabled (NEW_AND_OLD_IMAGES)
aws dynamodb create-table \
    --table-name MusicTable \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2

# Step 2: Create an identical MusicTable table in US East (N. Virginia)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "us-east-1"
      }
```

```
    }
  ]
}' \
--region us-east-2

# Step 3: Create a table in Europe (Ireland)
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Create": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2
```

Describa la tabla de varias regiones.

```
# Step 4: View the list of replicas created using describe-table
aws dynamodb describe-table \
    --table-name MusicTable \
    --region us-east-2 \
    --query 'Table.
{TableName:TableName,TableStatus:TableStatus,MultiRegionConsistency:MultiRegionConsistenc
{Region:RegionName,Status:ReplicaStatus}}'
```

Coloque elementos en una tabla de réplica.

```
# Step 5: To verify that replication is working, add a new item to the Music
 table in US East (Ohio)
aws dynamodb put-item \
    --table-name MusicTable \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

Obtenga elementos de las tablas de réplica.

```
# Step 6: Wait for a few seconds, and then check to see whether the item has
 been
# successfully replicated to US East (N. Virginia) and Europe (Ireland)
aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1

aws dynamodb get-item \
    --table-name MusicTable \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

Elimine réplicas.

```
# Step 7: Delete the replica table in Europe (Ireland) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "eu-west-1"
      }
    }
  ]
}' \
--region us-east-2

# Delete the replica table in US East (N. Virginia) Region
aws dynamodb update-table --table-name MusicTable --cli-input-json \
'{
  "ReplicaUpdates":
  [
    {
      "Delete": {
        "RegionName": "us-east-1"
      }
    }
  ]
}' \
--region us-east-2
```

Efectúe una limpieza eliminando la tabla.

```
# Clean up: Delete the primary table
aws dynamodb delete-table --table-name MusicTable --region us-east-2

echo "Global table demonstration complete."
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de comandos de AWS CLI.
  - CreateTable
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - UpdateTable

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Trabajo con el etiquetado de recursos de DynamoDB mediante AWS Command Line Interface v2

En el siguiente ejemplo de código se muestra cómo administrar etiquetas para los recursos de DynamoDB.

- Cree una tabla con etiquetas.
- Enumera las etiquetas de un recurso.
- Agregue etiquetas a un recurso.
- Elimine etiquetas de un recurso.
- Filtre tablas por etiquetas.

Bash

AWS CLI con Bash script

Cree una tabla con etiquetas.

```
# Create a table with tags
aws dynamodb create-table \
    --table-name TaggedTable \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
        Key=Environment,Value=Production \
        Key=Project,Value=Analytics \
        Key=Owner,Value=DataTeam
```

Enumera las etiquetas de un recurso.

```
# Get the table ARN
TABLE_ARN=$(aws dynamodb describe-table \
    --table-name TaggedTable \
    --query "Table.TableArn" \
    --output text)

# List tags for the table
aws dynamodb list-tags-of-resource \
    --resource-arn $TABLE_ARN
```

Agregue etiquetas a un recurso.

```
# Add tags to an existing table
aws dynamodb tag-resource \
    --resource-arn $TABLE_ARN \
    --tags \
        Key=CostCenter,Value=12345 \
        Key=BackupSchedule,Value=Daily
```

Elimine etiquetas de un recurso.

```
# Remove tags from a table
aws dynamodb untag-resource \
    --resource-arn $TABLE_ARN \
    --tag-keys Owner BackupSchedule
```

Filtre tablas por etiquetas.

```
# Create another table with different tags
aws dynamodb create-table \
    --table-name AnotherTaggedTable \
    --attribute-definitions \
        AttributeName=ID,AttributeType=S \
    --key-schema \
        AttributeName=ID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --tags \
        Key=Environment,Value=Development \
        Key=Project,Value=Testing

# Wait for table to become active
aws dynamodb wait table-exists --table-name AnotherTaggedTable

# List all tables
echo "All tables:"
aws dynamodb list-tables

# Get ARNs for all tables
echo -e "\nFiltering tables by Environment=Production tag:"
TABLE_ARNS=$(aws dynamodb list-tables --query "TableNames[*]" --output text |
 xargs -I {} aws dynamodb describe-table --table-name {} --query "Table.TableArn"
 --output text)

# Find tables with specific tag
for ARN in $TABLE_ARNS; do
    TABLE_NAME=$(echo $ARN | awk -F/ '{print $2}')
    TAGS=$(aws dynamodb list-tags-of-resource --resource-arn $ARN --query "Tags[?
Key=='Environment' && Value=='Production']" --output text)
    if [ ! -z "$TAGS" ]; then
        echo "Table with Production tag: $TABLE_NAME"
    fi
```

```
done
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de comandos de AWS CLI.
  - CreateTable
  - ListTagsOfResource
  - TagResource
  - UntagResource

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Trabajo con el cifrado de tablas de DynamoDB mediante AWS Command Line Interface v2

En el siguiente ejemplo de código se muestra cómo administrar las opciones de cifrado de las tablas de DynamoDB.

- Cree una tabla con un cifrado predeterminado.
- Cree una tabla con una CMK administrada por el cliente.
- Actualice la configuración de cifrado de la tabla.
- Describa el cifrado de la tabla.

Bash

AWS CLI con Bash script

Cree una tabla con un cifrado predeterminado.

```
# Create a table with default encryption (AWS owned key)
aws dynamodb create-table \
    --table-name CustomerData \
    --attribute-definitions \
        AttributeName=CustomerID,AttributeType=S \
    --key-schema \
```

```
        AttributeName=CustomerID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --sse-specification Enabled=true,SSEType=KMS
```

Cree una tabla con una CMK administrada por el cliente.

```
# Step 1: Create a customer managed key in KMS
aws kms create-key \
    --description "Key for DynamoDB table encryption" \
    --key-usage ENCRYPT_DECRYPT \
    --customer-master-key-spec SYMMETRIC_DEFAULT

# Store the key ID for later use
KEY_ID=$(aws kms list-keys --query "Keys[?contains(KeyArn, 'Key for
 DynamoDB')].KeyId" --output text)

# Step 2: Create a table with the customer managed key
aws dynamodb create-table \
    --table-name SensitiveData \
    --attribute-definitions \
        AttributeName=RecordID,AttributeType=S \
    --key-schema \
        AttributeName=RecordID,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=$KEY_ID
```

Actualice el cifrado de la tabla.

```
# Update a table to use a different KMS key
aws dynamodb update-table \
    --table-name CustomerData \
    --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=$KEY_ID
```

Describa el cifrado de la tabla.

```
# Describe the table to see encryption settings
aws dynamodb describe-table \
    --table-name CustomerData \
    --query "Table.SSEDescription"
```

- Para obtener información sobre la API, consulte los siguientes temas en la Referencia de comandos de AWS CLI.
  - CreateKey
  - CreateTable
  - DescribeTable
  - UpdateTable

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Ejemplos sin servidor para DynamoDB

En los siguientes ejemplos de código se muestra cómo utilizar DynamoDB con los SDK de AWS.

Ejemplos

- Invocación de una función de Lambda desde un desencadenador de DynamoDB
- Notificación de los errores de los elementos del lote de las funciones de Lambda con un desencadenador de DynamoDB

Invocación de una función de Lambda desde un desencadenador de DynamoDB

En los siguientes ejemplos de código se muestra cómo implementar una función de Lambda que recibe un evento desencadenado al recibir registros de una secuencia de DynamoDB. La función recupera la carga útil de DynamoDB y registra el contenido del registro.

.NET

SDK para .NET

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Consumo de un evento de DynamoDB con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext
 context)
    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
```

```
    }
}
```

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Consumo de un evento de DynamoDB con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-lambda-go/events"
 "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string,
 error) {
 if len(event.Records) == 0 {
  return nil, fmt.Errorf("received empty event")
 }

 for _, record := range event.Records {
   LogDynamoDBRecord(record)
 }

 message := fmt.Sprintf("Records processed: %d", len(event.Records))
 return &message, nil
}

func main() {
```

```
 lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
 fmt.Println(record.EventID)
 fmt.Println(record.EventName)
 fmt.Printf("%+v\n", record.Change)
}
```

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Consumo de un evento de DynamoDB con Lambda mediante Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import
 com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class example implements RequestHandler<DynamodbEvent, Void> {

    private static final Gson GSON = new
 GsonBuilder().setPrettyPrinting().create();

    @Override
    public Void handleRequest(DynamodbEvent event, Context context) {
        System.out.println(GSON.toJson(event));
        event.getRecords().forEach(this::logDynamoDBRecord);
        return null;
    }

    private void logDynamoDBRecord(DynamodbStreamRecord record) {
```

```
        System.out.println(record.getEventID());
        System.out.println(record.getEventName());
        System.out.println("DynamoDB Record: " +
 GSON.toJson(record.getDynamodb()));
    }
}
```

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Consumo de un evento de DynamoDB con Lambda mediante JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Consumo de un evento de DynamoDB con Lambda mediante TypeScript.

```
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
```

```
    });
}
const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Consumo de un evento de DynamoDB con Lambda mediante PHP.

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\DynamoDb\DynamoDbHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends DynamoDbHandler
{
    private StderrLogger $logger;

    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
```

```
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
    {
        $this->logger->info("Processing DynamoDb table items");
        $records = $event->getRecords();

        foreach ($records as $record) {
            $eventName = $record->getEventName();
            $keys = $record->getKeys();
            $old = $record->getOldImage();
            $new = $record->getNewImage();

            $this->logger->info("Event Name:".$eventName."\n");
            $this->logger->info("Keys:". json_encode($keys)."\n");
            $this->logger->info("Old Image:". json_encode($old)."\n");
            $this->logger->info("New Image:". json_encode($new));

            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }

        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords items");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Consumo de un evento de DynamoDB con Lambda mediante Python.

```
import json

def lambda_handler(event, context):
    print(json.dumps(event, indent=2))

    for record in event['Records']:
        log_dynamodb_record(record)

def log_dynamodb_record(record):
    print(record['eventID'])
    print(record['eventName'])
    print(f"DynamoDB Record: {json.dumps(record['dynamodb'])}")
```

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Consumo de un evento de DynamoDB con Lambda mediante Ruby.

```
def lambda_handler(event:, context:)
    return 'received empty event' if event['Records'].empty?

    event['Records'].each do |record|
      log_dynamodb_record(record)
    end

    "Records processed: #{event['Records'].length}"
  end

  def log_dynamodb_record(record)
```

```
    puts record['eventID']
    puts record['eventName']
    puts "DynamoDB Record: #{JSON.generate(record['dynamodb'])}"
  end
```

Rust

SDK para Rust

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Consumo de un evento de DynamoDB con Lambda mediante Rust.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord},
   };


// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) ->Result<(), Error> {

    let records = &event.payload.records;
    tracing::info!("event payload: {:?}",records);
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }
```

```
    for record in records{
        log_dynamo_dbrecord(record);
    }

    tracing::info!("Dynamo db records processed");

    // Prepare the response
    Ok(())

}

fn log_dynamo_dbrecord(record: &EventRecord)-> Result<(), Error>{
    tracing::info!("EventId: {}", record.event_id);
    tracing::info!("EventName: {}", record.event_name);
    tracing::info!("DynamoDB Record: {:?}", record.change );
    Ok(())

}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
    .with_max_level(tracing::Level::INFO)
    .with_target(false)
    .without_time()
    .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())

}
```

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Notificación de los errores de los elementos del lote de las funciones de Lambda con un desencadenador de DynamoDB

En los siguientes ejemplos de código se muestra cómo implementar una respuesta parcial por lotes para las funciones de Lambda que reciben eventos de una secuencia de DynamoDB. La función informa los errores de los elementos del lote en la respuesta y le indica a Lambda que vuelva a intentar esos mensajes más adelante.

.NET

SDK para .NET

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
 ILambdaContext context)

    {
```

```
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
 { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}
```

Go

SDK para Go V2

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

type BatchItemFailure struct {
 ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
 BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent)
 (*BatchResult, error) {
 var batchItemFailures []BatchItemFailure
 curRecordSequenceNumber := ""

 for _, record := range event.Records {
  // Process your record
  curRecordSequenceNumber = record.Change.SequenceNumber
 }

 if curRecordSequenceNumber != "" {
  batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
 curRecordSequenceNumber})
 }

 batchResult := BatchResult{
  BatchItemFailures: batchItemFailures,
 }

 return &batchResult, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Java

SDK para Java 2.x

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import com.amazonaws.services.lambda.runtime.events.models.dynamodb.StreamRecord;

import java.util.ArrayList;
import java.util.List;

public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
 StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(DynamodbEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (DynamodbEvent.DynamodbStreamRecord dynamodbStreamRecord :
 input.getRecords()) {
          try {
                //Process your record
```

```
                StreamRecord dynamodbRecord = dynamodbStreamRecord.getDynamodb();
                curRecordSequenceNumber = dynamodbRecord.getSequenceNumber();

            } catch (Exception e) {
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse();
    }
}
```

JavaScript

SDK para JavaScript (v3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante JavaScript.

```
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
```

```
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier:
 curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante TypeScript.

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
};
```

PHP

SDK para PHP

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante PHP.

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $dynamoDbEvent = new DynamoDbEvent($event);
        $this->logger->info("Processing records");

        $records = $dynamoDbEvent->getRecords();
```

```
        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK para Python (Boto3)

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["dynamodb"]["SequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDK para Ruby

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante Ruby.

```
def lambda_handler(event:, context:)
    records = event["Records"]
    cur_record_sequence_number = ""

    records.each do |record|
      begin
        # Process your record
```

```
        cur_record_sequence_number = record["dynamodb"]["SequenceNumber"]
      rescue StandardError => e
        # Return failed record's sequence number
        return {"batchItemFailures" => [{"itemIdentifier" =>
 cur_record_sequence_number}]}
      end
    end

    {"batchItemFailures" => []}
  end
```

Rust

SDK para Rust

> **Note**
>
> Hay más en GitHub. Busque el ejemplo completo y aprenda a configurar y ejecutar en el repositorio de ejemplos de tecnología sin servidor.

Notificación de los errores de los elementos del lote de DynamoDB con Lambda mediante Rust.

```
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord, StreamRecord},
    streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
    let stream_record: &StreamRecord = &record.change;

    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);

    Ok(())
}

/// Main Lambda handler here...
```

```
async fn function_handler(event: LambdaEvent<Event>) ->
 Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in records {
        tracing::info!("EventId: {}", record.event_id);

        // Couldn't find a sequence number
        if record.change.sequence_number.is_none() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: Some("".to_string()),
            });
            return Ok(response);
        }

        // Process your record here...
        if process_record(record).is_err() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: record.change.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }

    tracing::info!("Successfully processed {} record(s)", records.len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
```

```
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Contribuciones de la comunidad de AWS para DynamoDB

Las contribuciones de la comunidad de AWS son ejemplos creados y mantenidos por varios equipos de AWS. Para enviar comentarios, utilice el mecanismo previsto en los repositorios vinculados.

Ejemplos

- Cómo crear y probar una aplicación sin servidor

Cómo crear y probar una aplicación sin servidor

Los siguientes ejemplos de código muestran cómo crear y probar una aplicación sin servidor mediante API Gateway con Lambda y DynamoDB

.NET

SDK para .NET

Muestra cómo crear y probar una aplicación sin servidor que consta de una API Gateway con Lambda y DynamoDB mediante el SDK de .NET.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda

Go

SDK para Go V2

Muestra cómo crear y probar una aplicación sin servidor que consta de una API Gateway con Lambda y DynamoDB mediante el SDK de Go.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda

Java

SDK para Java 2.x

Muestra cómo crear y probar una aplicación sin servidor que consta de una API Gateway con Lambda y DynamoDB mediante el SDK de Java.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda

Rust

SDK para Rust

Muestra cómo crear y probar una aplicación sin servidor que consta de una API Gateway con Lambda y DynamoDB mediante el SDK de Rust.

Para ver el código fuente completo y las instrucciones de configuración y ejecución, consulte el ejemplo completo en GitHub.

Servicios utilizados en este ejemplo

- API Gateway
- DynamoDB
- Lambda

Para obtener una lista completa de las guías para desarrolladores de AWS SDK y ejemplos de código, consulte Uso de DynamoDB con un SDK de AWS. En este tema también se incluye información sobre cómo comenzar a utilizar el SDK y detalles sobre sus versiones anteriores.

Seguridad y conformidad en Amazon DynamoDB

La seguridad en la nube de AWS es la mayor prioridad. Como cliente de AWS, se beneficia de una arquitectura de red y un centro de datos que se han diseñado para satisfacer los requisitos de seguridad de las organizaciones más exigentes.

La seguridad es una responsabilidad compartida entre AWS y el usuario. El modelo de responsabilidad compartida la describe como seguridad de la nube y seguridad en la nube:

- Seguridad de la nube: AWS es responsable de proteger la infraestructura que ejecuta los servicios de AWS en la nube de AWS AWS también proporciona servicios que puede utilizar de forma segura. Auditores externos prueban y verifican periódicamente la eficacia de nuestra seguridad en el marco de los programas de conformidad de AWS. Para obtener más información acerca de los programas de conformidad que se aplican a DynamoDB, consulte Servicios de AWS en el ámbito del programa de conformidad.
- Seguridad en la nube: su responsabilidad viene determinada por el servicio de AWS que utilice. Usted también es responsable de otros factores, incluida la confidencialidad de los datos, los requisitos de la empresa y la legislación y los reglamentos aplicables.

Esta documentación lo ayudará a comprender cómo aplicar el modelo de responsabilidad compartida cuando se utiliza DynamoDB. En los siguientes temas, se le mostrará cómo configurar DynamoDB para satisfacer sus objetivos de seguridad y conformidad. También puede aprender a utilizar otros servicios de AWS que le ayudan a monitorear y proteger sus recursos de DynamoDB.

Temas

- Políticas administradas por AWS para Amazon DynamoDB
- Uso de políticas basadas en recursos para DynamoDB
- Uso del control de acceso basado en atributos con DynamoDB
- Protección de datos en DynamoDB
- AWS Identity and Access Management (IAM) y DynamoDB
- Validación de conformidad por sector para DynamoDB
- Resiliencia y recuperación de desastres en Amazon DynamoDB
- Seguridad de la infraestructura en Amazon DynamoDB
- AWS PrivateLink para DynamoDB

- Configuración y análisis de vulnerabilidades en Amazon DynamoDB
- Prácticas recomendadas de seguridad para Amazon DynamoDB

Políticas administradas por AWS para Amazon DynamoDB

DynamoDB utiliza las políticas administradas por AWS para definir un conjunto de permisos que el servicio necesita a fin de realizar acciones específicas. DynamoDB mantiene y actualiza sus políticas administradas por AWS. No puede cambiar los permisos en las políticas gestionadas de AWS. Para obtener más información sobre las políticas administradas por AWS, consulte Políticas administradas porAWS en la guía del usuario de IAM.

En ocasiones, DynamoDB puede agregar permisos adicionales a una política administrada por AWS para admitir características nuevas. Este tipo de actualización afecta a todas las identidades (usuarios, grupos y roles) donde se asocia la política. Lo más probable es que una política administrada por AWS se actualice cuando se lance una nueva característica o cuando haya nuevas operaciones disponibles. DynamoDB no eliminará los permisos de una política administrada por AWS, por lo que las actualizaciones de la política no anularán sus permisos existentes. Para obtener una lista completa de las políticas administradas de AWS, consulte Políticas administradas de AWS.

Política administrada por AWS: DynamoDBReplicationServiceRolePolicy

No puede adjuntar la política `DynamoDBReplicationServiceRolePolicy` a sus entidades de IAM. Esta política está adjuntada a un rol vinculado a servicios que permite a DynamoDB realizar acciones en su nombre. Para obtener más información, consulte Uso de IAM con tablas globales.

Esta política concede permisos que permiten al rol vinculado a servicios realizar la replicación de datos entre réplicas de tablas globales. También concede permisos administrativos para administrar las réplicas de tablas globales en su nombre.

Detalles de los permisos

Esta política concede permisos para hacer lo siguiente:

- `dynamodb`: realizar la replicación de datos y administrar las réplicas de tablas.
- `application-autoscaling`: recuperar y administrar la configuración de escalado automático de la tabla.
- `account`: recuperar el estado de la región para evaluar la accesibilidad de la réplica.

- `iam`: para crear el rol vinculado a servicios para el escalado automático de aplicaciones en caso de que dicho rol no exista todavía.

La definición de esta política administrada se encuentra aquí.

Política administrada de AWS: AmazonDynamoDBFullAccess_v2

La política `AmazonDynamoDBFullAccess_v2` de ámbito reducido concede privilegios de acceso específicos a los usuarios. Puede adjuntar la política `AmazonDynamoDBFullAccess_v2` a las identidades de IAM. Esta política concede acceso administrativo a los recursos de Amazon DynamoDB y otorga a una identidad de IAM (como un usuario, grupo o rol) acceso a los Servicios de AWS con los que DynamoDB está integrado para utilizar todas las características de DynamoDB. El uso de esta política permite acceder a todas las características de DynamoDB que están disponibles en la Consola de administración de AWS.

Detalles de los permisos

Esta política incluye los permisos siguientes:

- `Amazon DynamoDB`
- `DynamoDB Accelerator`
- `AWS KMS`
- `Grupos de recursos de AWS Tagging`
- `Lambda`
- `Application Auto Scaling`
- `CloudWatch`
- `Amazon Kinesis`
- `Amazon EC2`
- `IAM`

Para revisar la política en formato `JSON`, consulte AmazonDynamoDBFullAccess_v2.

Política administrada de AWS: AmazonDynamodBreadOnlyAccess

Puede adjuntar la política `AmazonDynamoDBReadOnlyAccess` a las identidades de IAM.

Esta política concede acceso de lectura a Amazon DynamoDB.

Detalles de los permisos

Esta política incluye los permisos siguientes:

- `Amazon DynamoDB`: proporciona acceso de solo lectura a Amazon DynamoDB.
- `Amazon DynamoDB Accelerator (DAX)`: proporciona acceso de solo lectura a Acelerador de Amazon DynamoDB (DAX).
- `Application Auto Scaling`: permite a las entidades principales ver las configuraciones de Application Auto Scaling. Esto es necesario para que los usuarios puedan ver las políticas de escalado automático que se vinculan a una tabla.
- `CloudWatch`: permite a las entidades principales ver los datos métricos y las alarmas configuradas en CloudWatch. Esto es necesario para que los usuarios puedan ver el tamaño facturable de la tabla y las alarmas de CloudWatch que se han configurado para una tabla.
- `AWS Data Pipeline`: permite a las entidades principales ver AWS Data Pipeline y los objetos asociados.
- `Amazon EC2`: permite a las entidades principales ver las VPC, las subredes y los grupos de seguridad de Amazon EC2.
- `IAM`: permite a las entidades principales ver los roles de IAM.
- `AWS KMS`: permite a las entidades principales ver las claves configuradas en AWS KMS. Esto es necesario para que los usuarios puedan ver las AWS KMS keys que crean y administran en sus cuentas.
- `Amazon SNS`: permite a las entidades principales mostrar temas de Amazon SNS y suscripciones por tema.
- `Grupos de recursos de AWS`: permite a las entidades principales ver grupos de recursos y sus consultas.
- `Grupos de recursos de AWS Tagging`: permite a las entidades principales enumerar todos los recursos etiquetas o etiquetados previamente en una región.
- `Kinesis`: permite a las entidades principales ver las descripciones de los flujos de datos de Kinesis.
- `Amazon CloudWatch Contributor Insights`: permite a las entidades principales ver los datos de serie temporal recopilados por las reglas de Contributor Insights.

Para revisar la política en formato `JSON`, consulte AmazonDynamoDBReadOnlyAccess.

Actualizaciones de DynamoDB en las políticas administradas por AWS

En esta tabla se muestran las actualizaciones de las políticas de administración de acceso de AWS para DynamoDB.

Cambio	Descripción	Fecha de modificación
`AmazonDynamoDBFullAccess` : obsoleta	Esta política se ha reemplazado por una política de ámbito reducido denominada `AmazonDynamoDBFullAccess_v2` . Después de abril de 2025, no podrá asociar la política `AmazonDynamoDBFullAccess` a ningún usuario, grupo o rol nuevo. Para obtener más información, consulte Política administrada de AWS: AmazonDynamoDBFullAccess_v2.	28 de abril de 2025
`AmazonDynamoDBReadOnlyAccess` Actualización de a una política existente	`AmazonDynamoDBReadOnlyAccess` agregó los permisos `dynamodb:GetAbacStatus` y `dynamodb:UpdateAbacStatus` . Estos permisos le permiten ver el estado de ABAC y habilitar ABAC para la Cuenta de AWS en la región actual.	18 de noviembre de 2024
`AmazonDynamoDBReadOnlyAccess` Actualización de a una política existente	`AmazonDynamoDBReadOnlyAccess` agregó el permiso `dynamodb:GetResourcePolicy` . Este permiso proporciona acceso a políticas basadas en recursos de lectura asociadas a los recursos de DynamoDB.	20 de marzo de 2024

Cambio	Descripción	Fecha de modificación
Actualización de `DynamoDBReplicationServiceRolePolicy` a una política existente	`DynamoDBReplicationServiceRolePolicy` agregó el permiso `dynamodb:GetResourcePolicy`. Permite al rol vinculado al servicio leer las políticas basadas en recursos asociadas a los recursos de DynamoDB.	15 de diciembre de 2023
Actualización de `DynamoDBReplicationServiceRolePolicy` a una política existente	`DynamoDBReplicationServiceRolePolicy` agregó el permiso `account:ListRegions`. Este permiso permite al rol vinculado a servicios evaluar la accesibilidad de la réplica	10 de mayo de 2023
Se agregó `DynamoDBReplicationServiceRolePolicy` a la lista de políticas administradas	Se agregó información sobre la política administrada `DynamoDBReplicationServiceRolePolicy`, que utiliza el rol vinculado a servicios de tablas globales de DynamoDB.	10 de mayo de 2023
Las tablas globales de DynamoDB empiezan a hacer un seguimiento de los cambios	Las tablas globales de DynamoDB comenzaron a hacer un seguimiento de los cambios de sus políticas administradas por AWS.	10 de mayo de 2023

Uso de políticas basadas en recursos para DynamoDB

DynamoDB admite las políticas basadas en recursos para tablas, índices y secuencias. Las políticas basadas en recursos permiten definir los permisos de acceso al especificar quién tiene acceso a cada recurso y las acciones que puede realizar en cada recurso.

Puede asociar una política basada en recursos a los recursos de DynamoDB, como una tabla o una secuencia. En esta política, se especifican los permisos para las entidades principales de Identity and Access Management (IAM) que pueden realizar acciones específicas en estos recursos de DynamoDB. Por ejemplo, la política asociada a una tabla incluirá los permisos de acceso a la tabla y a sus índices. Como resultado, las políticas basadas en recursos pueden ayudarlo a simplificar el control de acceso a las tablas, índices y secuencias de DynamoDB, al definir los permisos en el nivel de recursos. El tamaño máximo de una política que puede adjuntar a un recurso de DynamoDB es de 20 KB.

Una ventaja importante del uso de políticas basadas en recursos es que simplifica el control de acceso entre cuentas para proporcionar acceso entre cuentas a las entidades principales de IAM en diferentes Cuentas de AWS. Para obtener más información, consulte Políticas basadas en recursos para el acceso entre cuentas.

Las políticas basadas en los recursos también admiten las integraciones con el analizador de acceso externo de IAM Access Analyzer y las funciones Bloquear el acceso público (BPA). IAM Access Analyzer informa sobre el acceso entre cuentas a entidades externas especificadas en las políticas basadas en recursos. También proporciona visibilidad para ayudarlo a refinar los permisos y cumplir con el principio de privilegios mínimos. El BPA lo ayuda a impedir el acceso público a las tablas, índices y secuencias de DynamoDB. Además, se activa automáticamente en los flujos de trabajo de creación y modificación de las políticas basadas en recursos.

Temas

- Creación de una tabla con una política basada en recursos
- Asociación de una política a una tabla de DynamoDB existente
- Asociación de una política basada en recursos a un flujo de DynamoDB
- Eliminación de una política basada en recursos de una tabla de DynamoDB
- Acceso entre cuentas con políticas basadas en recursos en DynamoDB
- Bloqueo del acceso público con políticas basadas en recursos en DynamoDB
- Operaciones de la API de DynamoDB admitidas por las políticas basadas en recursos

- Autorización con políticas basadas en identidad de IAM y políticas basadas en recursos de DynamoDB
- Ejemplos de política basada en recursos de DynamoDB
- Consideraciones sobre la política basada en recursos de DynamoDB
- Prácticas recomendadas de las políticas basadas en recursos de DynamoDB

Creación de una tabla con una política basada en recursos

Puede agregar una política basada en recursos al crear una tabla desde la consola de DynamoDB, la API CreateTable, la AWS CLI, el AWS SDK o una plantilla de CloudFormation.

AWS CLI

En el siguiente ejemplo se crea una tabla denominada *MusicCollection* con el comando `create-table` de la AWS CLI. Este comando también incluye el parámetro `resource-policy` que añade a la tabla una política basada en recursos. Esta política permite al usuario *John* realizar las acciones de las API RestoreTableToPointInTime, GetItem y PutItem en la tabla.

Recuerde reemplazar el texto en *cursiva* por la información específica del recurso.

```
aws dynamodb create-table \
    --table-name MusicCollection \
    --attribute-definitions AttributeName=Artist,AttributeType=S
 AttributeName=SongTitle,AttributeType=S \
    --key-schema AttributeName=Artist,KeyType=HASH
 AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --resource-policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::123456789012:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:RestoreTableToPointInTime\",
                        \"dynamodb:GetItem\",
                        \"dynamodb:DescribeTable\"
```

```
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
            ]
        }"
```

Consola de administración de AWS

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel, elija Crear tabla.
3. En Detalles de la tabla, especifique el nombre de la tabla, la clave de partición y los detalles de la clave de clasificación.
4. En Configuración de la tabla, elija Personalizar configuración.
5. (Opcional) Especifique sus opciones para Clase de tabla, Calculadora de capacidad, Configuración de capacidad de lectura/escritura, Índices secundarios, Cifrado en reposo y Protección contra eliminaciones.
6. En Política basada en recursos, añada una política para definir los permisos de acceso a la tabla y sus índices. En esta política, se especifica quién tiene acceso a estos recursos y las acciones que se les permite realizar en cada recurso. Para añadir una política, realice alguna de las siguientes operaciones:

   - Escriba o pegue un documento de política de JSON. Para obtener más información sobre el lenguaje de las políticas de IAM, consulte Creación de políticas mediante el editor JSON en la Guía del usuario de IAM.

     > **Tip**
     >
     > Para ver ejemplos de políticas basadas en recursos en la Guía para desarrolladores de Amazon DynamoDB, elija Ejemplos de políticas.

   - Elija Agregar nueva instrucción. A continuación, añada una nueva instrucción y especifique la información en los campos proporcionados. Repita este paso para tantas instrucciones como desee agregar.

> **Important**
>
> Asegúrese de resolver advertencias de seguridad, los errores y las sugerencias antes de guardar la política.

La siguiente política de IAM de ejemplo permite al usuario *John* realizar las acciones de la API RestoreTableToPointInTime, GetItem y PutItem en la tabla *MusicCollection*.

Recuerde reemplazar el texto en *cursiva* por la información específica del recurso.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/username"
      },
      "Action": [
        "dynamodb:RestoreTableToPointInTime",
        "dynamodb:GetItem",
        "dynamodb:PutItem"
      ],
      "Resource": "arn:aws:dynamodb:us-
east-1:123456789012:table/MusicCollection"
    }
  ]
}
```

7. (Opcional) Elija Preview external access (Vista previa del acceso externo) en la esquina inferior derecha para obtener una vista previa de cómo la política nueva afecta al acceso público y entre cuentas al recurso. Antes de guardar la política, puede comprobar si introduce nuevos hallazgos de IAM Access Analyzer o resuelve las conclusiones existentes. Si no ve un analizador activo, elija Ir a Access Analyzer para crear un analizador de la cuenta en Access Analyzer de IAM. Para obtener más información, consulte Acceso de vista previa.

8. Seleccione Create table (Creación de tabla).

AWS CloudFormationPlantilla de

Using the AWS::DynamoDB::Table resource

La siguiente plantilla de CloudFormation crea una tabla con una secuencia mediante el recurso AWS::DynamoDB::Table. Esta plantilla también incluye políticas basadas en recursos que se adjuntan tanto a la tabla como a la secuencia.

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "MusicCollectionTable": {
            "Type": "AWS::DynamoDB::Table",
            "Properties": {
                "AttributeDefinitions": [
                    {
                        "AttributeName": "Artist",
                        "AttributeType": "S"
                    }
                ],
                "KeySchema": [
                    {
                        "AttributeName": "Artist",
                        "KeyType": "HASH"
                    }
                ],
                "BillingMode": "PROVISIONED",
                "ProvisionedThroughput": {
                    "ReadCapacityUnits": 5,
                    "WriteCapacityUnits": 5
                },
                "StreamSpecification": {
                  "StreamViewType": "OLD_IMAGE",
                  "ResourcePolicy": {
                    "PolicyDocument": {
                      "Version": "2012-10-17",
                      "Statement": [
                        {
                            "Principal": {
                                "AWS": "arn:aws:iam::111122223333:user/John"
                            },
```

```
                            "Effect": "Allow",
                            "Action": [
                                "dynamodb:GetRecords",
                                "dynamodb:GetShardIterator",
                                "dynamodb:DescribeStream"
                            ],
                            "Resource": "*"
                        }
                    ]
                }
            }
        },
        "TableName": "MusicCollection",
        "ResourcePolicy": {
            "PolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Principal": {
                            "AWS": [
                                "arn:aws:iam::111122223333:user/John"
                            ]
                        },
                        "Effect": "Allow",
                        "Action": "dynamodb:GetItem",
                        "Resource": "*"
                    }
                ]
            }
        }
    }

        }
    }
}
```

Using the AWS::DynamoDB::GlobalTable resource

La siguiente plantilla de CloudFormation crea una tabla con el recurso AWS::DynamoDB::GlobalTable y adjunta una política basada en recursos a la tabla y a su secuencia.

```
{
```

```
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "GlobalMusicCollection": {
            "Type": "AWS::DynamoDB::GlobalTable",
            "Properties": {
                "TableName": "MusicCollection",
                "AttributeDefinitions": [{
                    "AttributeName": "Artist",
                    "AttributeType": "S"
                }],
                "KeySchema": [{
                    "AttributeName": "Artist",
                    "KeyType": "HASH"
                }],
                "BillingMode": "PAY_PER_REQUEST",
                "StreamSpecification": {
                    "StreamViewType": "NEW_AND_OLD_IMAGES"
                },
                "Replicas": [
                    {
                        "Region": "us-east-1",
                        "ResourcePolicy": {
                            "PolicyDocument": {
                                "Version": "2012-10-17",
                                "Statement": [{
                                    "Principal": {
                                        "AWS": [
                                            "arn:aws:iam::111122223333:user/John"
                                        ]
                                    },
                                    "Effect": "Allow",
                                    "Action": "dynamodb:GetItem",
                                    "Resource": "*"
                                }]
                            }
                        },
                        "ReplicaStreamSpecification": {
                            "ResourcePolicy": {
                                "PolicyDocument": {
                                    "Version": "2012-10-17",
                                    "Statement": [{
                                        "Principal": {
                                            "AWS":
"arn:aws:iam::111122223333:user/John"
```

```
                                    },
                                    "Effect": "Allow",
                                    "Action": [
                                        "dynamodb:GetRecords",
                                        "dynamodb:GetShardIterator",
                                        "dynamodb:DescribeStream"
                                    ],
                                    "Resource": "*"
                                }]
                            }
                        }
                    }
                ]
            }
        }
    }
}
```

Asociación de una política a una tabla de DynamoDB existente

Puede asociar una política basada en recursos a una tabla existente o modificar una política existente desde la consola de DynamoDB, la API PutResourcePolicy, la AWS CLI, el SDK de AWS o la plantilla de CloudFormation.

Ejemplo de AWS CLI para asociar una nueva política

En el siguiente ejemplo de política de IAM se utiliza el comando `put-resource-policy` de la AWS CLI para asociar una política basada en recursos a una tabla existente. En este ejemplo, el usuario *`John`* puede ejecutar las acciones de la API GetItem, PutItem, UpdateItem y UpdateTable en una tabla existente denominada *`MusicCollection`*.

Recuerde reemplazar el texto en *`cursiva`* por la información específica del recurso.

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
```

```
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:GetItem\",
                        \"dynamodb:PutItem\",
                        \"dynamodb:UpdateItem\",
                        \"dynamodb:UpdateTable\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
            ]
        }"
```

Ejemplo de AWS CLI para actualizar condicionalmente una política existente

Para actualizar condicionalmente una política basada en recursos existente de una tabla, puede usar el parámetro `expected-revision-id` opcional. En el siguiente ejemplo solo se actualiza la política si existe en DynamoDB y si su ID de revisión actual coincide con el parámetro `expected-revision-id` proporcionado.

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --expected-revision-id 1709841168699 \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:GetItem\",
                        \"dynamodb:UpdateItem\",
                        \"dynamodb:UpdateTable\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection\"
                }
```

```
        ]
    }"
```

Consola de administración de AWS

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de, elija una tabla existente.
3. Vaya a la pestaña Permisos y elija Crear política de tabla.
4. En el editor de políticas basadas en recursos, añada la política que desea asociar y elija Crear política.

   En el siguiente ejemplo de política de IAM, el usuario *John* puede ejecutar las acciones de la API GetItem, PutItem, UpdateItem y UpdateTable en una tabla existente denominada *MusicCollection*.

   Recuerde reemplazar el texto en *cursiva* por la información específica del recurso.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/username"
      },
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:UpdateTable"
      ],
      "Resource": "arn:aws:dynamodb:us-
east-1:123456789012:table/MusicCollection"
    }
  ]
}
```

AWS SDK for Java 2.x

En el siguiente ejemplo de política de IAM se utiliza el método `putResourcePolicy` para asociar una política basada en recursos a una tabla existente. Esta política permite a un usuario realizar la acción de la API GetItem en una tabla existente.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbClient;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
import software.amazon.awssdk.services.dynamodb.model.PutResourcePolicyRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * Get started with the AWS SDK for Java 2.x
 */
public class PutResourcePolicy {

    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <tableArn> <allowedAWSPrincipal>

                Where:
                    tableArn - The Amazon DynamoDB table ARN to attach the policy to.
 For example, arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection.
                    allowedAWSPrincipal - Allowed AWS principal ARN that the example
 policy will give access to. For example, arn:aws:iam::123456789012:user/John.
                """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String tableArn = args[0];
        String allowedAWSPrincipal = args[1];
        System.out.println("Attaching a resource-based policy to the Amazon DynamoDB
 table with ARN " +
```

```
                tableArn);
        Region region = Region.US_WEST_2;
        DynamoDbClient ddb = DynamoDbClient.builder()
                .region(region)
                .build();

        String result = putResourcePolicy(ddb, tableArn, allowedAWSPrincipal);
        System.out.println("Revision ID for the attached policy is " + result);
        ddb.close();
    }

    public static String putResourcePolicy(DynamoDbClient ddb, String tableArn, String
 allowedAWSPrincipal) {
        String policy = generatePolicy(tableArn, allowedAWSPrincipal);
        PutResourcePolicyRequest request = PutResourcePolicyRequest.builder()
                .policy(policy)
                .resourceArn(tableArn)
                .build();

        try {
            return ddb.putResourcePolicy(request).revisionId();
        } catch (DynamoDbException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }

        return "";
    }

    private static String generatePolicy(String tableArn, String allowedAWSPrincipal) {
        return "{\n" +
                "    \"Version\": \"2012-10-17\",\n" +,
                "    \"Statement\": [\n" +
                "        {\n" +
                "            \"Effect\": \"Allow\",\n" +
                "            \"Principal\": {\"AWS\":\"" + allowedAWSPrincipal + "\"},
\n" +
                "            \"Action\": [\n" +
                "                \"dynamodb:GetItem\"\n" +
                "            ],\n" +
                "            \"Resource\": \"" + tableArn + "\"\n" +
                "        }\n" +
                "    ]\n" +
                "}";
```

```
    }
}
```

Asociación de una política basada en recursos a un flujo de DynamoDB

Puede asociar una política basada en recursos a una secuencia de tablas existente o modificar una política existente desde la consola de DynamoDB, la API PutResourcePolicy, la AWS CLI, el SDK de AWS o la plantilla de CloudFormation.

> **Note**
>
> No puede asociar una política a una secuencia mientras se crea con las API CreateTable o UpdateTable. Sin embargo, puede modificar o eliminar una política después de eliminar una tabla. También puede modificar o eliminar la política de una secuencia desactivada.

AWS CLI

En el siguiente ejemplo de política de IAM se utiliza el comando `put-resource-policy` de la AWS CLI para asociar una política basada en recursos a una secuencia de una tabla denominada *`MusicCollection`*. En este ejemplo, el usuario *`John`* puede realizar las acciones de las API GetRecords, GetShardIterator y DescribeStream en la secuencia.

Recuerde reemplazar el texto en *`cursiva`* por la información específica del recurso.

```
aws dynamodb put-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492 \
    --policy \
        "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
              {
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"AWS\": \"arn:aws:iam::111122223333:user/John\"
                    },
                    \"Action\": [
                        \"dynamodb:GetRecords\",
```

```
                        \"dynamodb:GetShardIterator\",
                        \"dynamodb:DescribeStream\"
                    ],
                    \"Resource\": \"arn:aws:dynamodb:us-
west-2:123456789012:table/MusicCollection/stream/2024-02-12T18:57:26.492\"
                }
            ]
        }"
```

Consola de administración de AWS

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de la consola de DynamoDB, elija Tablas y seleccione una tabla existente.

   Asegúrese de que la tabla que seleccione tenga las secuencias activadas. Para obtener información sobre la activación de las secuencias en una tabla, consulte Habilitación de una secuencia.
3. Elija la pestaña Permisos.
4. En Política basada en recursos para la secuencia activa, elija Crear política de transmisión.
5. En el editor Política basada en recursos, añada una política para definir los permisos de acceso a la secuencia. En esta política, se especifica quién tiene acceso a la secuencia y las acciones que se les permite realizar en cada secuencia. Para añadir una política, realice alguna de las siguientes operaciones:

   - Escriba o pegue un documento de política de JSON. Para obtener más información sobre el lenguaje de las políticas de IAM, consulte Creación de políticas mediante el editor JSON en la Guía del usuario de IAM.

     > **Tip**
     >
     > Para ver ejemplos de políticas basadas en recursos en la Guía para desarrolladores de Amazon DynamoDB, elija Ejemplos de políticas.

   - Elija Agregar nueva instrucción. A continuación, añada una nueva instrucción y especifique la información en los campos proporcionados. Repita este paso para tantas instrucciones como desee agregar.

> **Important**
>
> Asegúrese de resolver advertencias de seguridad, los errores y las sugerencias antes de guardar la política.

6. (Opcional) Elija Preview external access (Vista previa del acceso externo) en la esquina inferior derecha para obtener una vista previa de cómo la política nueva afecta al acceso público y entre cuentas al recurso. Antes de guardar la política, puede comprobar si introduce nuevos hallazgos de IAM Access Analyzer o resuelve las conclusiones existentes. Si no ve un analizador activo, elija Ir a Access Analyzer para crear un analizador de la cuenta en Access Analyzer de IAM. Para obtener más información, consulte Acceso de vista previa.
7. Seleccione Crear política.

En el siguiente ejemplo de política de IAM se asocia una política basada en recursos a una secuencia de una tabla denominada *MusicCollection*. En este ejemplo, el usuario *John* puede realizar las acciones de las API GetRecords, GetShardIterator y DescribeStream en la secuencia.

Recuerde reemplazar el texto en *cursiva* por la información específica del recurso.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/username"
      },
      "Action": [
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator",
        "dynamodb:DescribeStream"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492"
      ]
```

```
    }
  ]
}
```

Eliminación de una política basada en recursos de una tabla de DynamoDB

Puede eliminar una política basada en recursos de una tabla existente desde la consola de DynamoDB, la API DeleteResourcePolicy, la AWS CLI, el SDK de AWS o una plantilla de CloudFormation.

AWS CLI

En el siguiente ejemplo se utiliza el comando `delete-resource-policy` de la AWS CLI para eliminar una política basada en recursos de una tabla denominada *`MusicCollection`*.

Recuerde reemplazar el texto en *`cursiva`* por la información específica del recurso.

```
aws dynamodb delete-resource-policy \
    --resource-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection
```

Consola de administración de AWS

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de la consola de DynamoDB, elija Tablas y seleccione una tabla existente.
3. Elija Permisos.
4. En el menú desplegable Administrar política, seleccione Eliminar política.
5. En el cuadro de diálogo Eliminar política basada en recursos de la tabla, escriba **`confirm`** para confirmar la acción de eliminación.
6. Elija Eliminar.

Acceso entre cuentas con políticas basadas en recursos en DynamoDB

Al usar una política basada en recursos, puede proporcionar acceso entre cuentas a los recursos disponibles en diferentes Cuentas de AWS. Todos los accesos entre cuentas permitidos por las políticas basadas en recursos se reflejarán en los resultados del acceso externo de IAM Access Analyzer si dispone de un analizador en la misma Región de AWS que el recurso. IAM

Access Analyzer ejecuta verificaciones de política para validarla contra la Gramática de la política de IAM y las prácticas recomendadas. Estas verificaciones generan hallazgos y proporcionan recomendaciones procesables para ayudarlo a crear políticas funcionales y que se ajustan a las prácticas recomendadas de seguridad. Puede ver los resultados activos de IAM Access Analyzer en la pestaña Permisos de la consola de DynamoDB.

Para obtener más información sobre la validación de políticas con IAM Access Analyzer, consulte Política de validación de Analizador de acceso de IAM en la Guía del usuario de IAM. Para ver una lista de advertencias, errores y sugerencias que devuelve IAM Access Analyzer, consulte Referencia de verificación de políticas de IAM Access Analyzer.

Para conceder el permiso GetItem a un usuario A de la cuenta A para acceder a una tabla B de la cuenta B, lleve a cabo los siguientes pasos:

1. Asocie una política basada en recursos a la tabla B que otorgue permiso al usuario A para realizar la acción `GetItem`.
2. Asocie una política basada en identidad a un usuario A que le permita realizar la acción `GetItem` en la tabla B.

Si la opción Vista previa del acceso externo está disponible en la consola de DynamoDB, podrá ver cómo afectará al recurso la nueva política de acceso público y entre cuentas. Antes de guardar la política, puede comprobar si introduce nuevos hallazgos de IAM Access Analyzer o resuelve las conclusiones existentes. Si no ve un analizador activo, elija Ir a Access Analyzer para crear un analizador de la cuenta en Access Analyzer de IAM. Para obtener más información, consulte Acceso de vista previa.

El parámetro de nombre de tabla de las API de plano de datos y plano de control de DynamoDB aceptan el Nombre de recurso de Amazon (ARN) completo de la tabla para admitir las operaciones entre cuentas. Si solo proporciona el parámetro de nombre de tabla en lugar de un ARN completo, la operación de la API se realizará en la tabla de la cuenta a la que pertenece el solicitante. Para ver un ejemplo de una política que usa el acceso entre cuentas, consulte Políticas basadas en recursos para el acceso entre cuentas.

Se cobrará a la cuenta del propietario del recurso incluso cuando la entidad principal de otra cuenta lea o escriba en la tabla de DynamoDB de la cuenta del propietario. Si la tabla tiene un rendimiento aprovisionado, la suma de todas las solicitudes de las cuentas del propietario y de los solicitantes de otras cuentas determinará si se limitará la solicitud (si el escalado automático está desactivado) o si se escalará o reducirá verticalmente si el escalado automático está activado.

Las solicitudes se registrarán en los registros de CloudTrail de las cuentas del propietario y del solicitante para que cada una de las dos cuentas pueda rastrear qué cuenta accedió a qué datos.

> **Note**
>
> El acceso entre cuentas a las API de plano de control tiene un límite inferior de 500 solicitudes de transacciones por segundo (TPS).

Bloqueo del acceso público con políticas basadas en recursos en DynamoDB

Bloquear el acceso público (BPA) es una característica que identifica y evita que se asocien políticas basadas en recursos que concedan acceso público a sus tablas, índices o secuencias de DynamoDB en sus cuentas de Amazon Web Services (AWS). Con BPA, puede impedir el acceso público a sus recursos de DynamoDB. BPA realiza comprobaciones durante la creación o modificación de una política basada en recursos y ayuda a mejorar la seguridad con DynamoDB.

BPA utiliza un razonamiento automatizado para analizar el acceso que concede su política basada en recursos y le avisa si encuentra dichos permisos al administrar una política basada en recursos. El análisis verifica el acceso a todas las instrucciones de políticas basadas en recursos, las acciones y el conjunto de claves de condición que se utilizan en sus políticas.

> **Important**
>
> BPA ayuda a proteger sus recursos al impedir que se conceda acceso público a través de las políticas basadas en recursos que se asocian directamente a sus recursos de DynamoDB, como tablas, índices y secuencias. Además de habilitar BPA, analice detenidamente las siguientes políticas para confirmar que no otorgan acceso público:
>
> - Políticas basadas en identidad asociadas a las entidades principales de AWS vinculadas (por ejemplo, los roles de IAM)
> - Políticas basadas en recursos asociadas a recursos de AWS vinculados (por ejemplo, claves AWS Key Management Service [KMS])

Debe asegurarse de que la entidad principal no incluya una entrada * o que una de las claves de condición especificadas restrinja el acceso de las entidades principales al recurso. Si la política

basada en recursos concede acceso público a la tabla, los índices o la secuencia en las Cuentas de AWS, DynamoDB le impedirá crear o modificar la política hasta que se corrija la especificación de la política y se considere no pública.

Para que una política no sea pública, especifique una o más entidades principales dentro del bloque `Principal`. El siguiente ejemplo de política basada en recursos bloquea el acceso público al especificar dos entidades principales.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "123456789012",
      "111122223333"
    ]
  },
  "Action": "dynamodb:*",
  "Resource": "*"
}
```

Las políticas que restringen el acceso al especificar determinadas claves de condición tampoco se consideran públicas. Junto con la evaluación de la entidad principal especificada en la política basada en recursos, se utilizan las siguientes claves de condición fiables para completar la evaluación de una política basada en recursos para el acceso no público:

- `aws:PrincipalAccount`
- `aws:PrincipalArn`
- `aws:PrincipalOrgID`
- `aws:PrincipalOrgPaths`
- `aws:SourceAccount`
- `aws:SourceArn`
- `aws:SourceVpc`
- `aws:SourceVpce`
- `aws:UserId`
- `aws:PrincipalServiceName`
- `aws:PrincipalServiceNamesList`
- `aws:PrincipalIsAWSService`

- aws:Ec2InstanceSourceVpc
- aws:SourceOrgID
- aws:SourceOrgPaths

Además, para que una política basada en recursos no sea pública, los valores del Nombre de recurso de Amazon (ARN) y las claves de cadena no deben contener comodines ni variables. Si su política basada en recursos usa la clave aws:PrincipalIsAWSService, debe asegurarse de establecer el valor de la clave en true.

La siguiente política limita el acceso al usuario John de la cuenta especificada. La condición limita a la Principal y no se considera pública.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "*"
  },
  "Action": "dynamodb:*",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:PrincipalArn": "arn:aws:iam::123456789012:user/John"
    }
  }
}
```

El siguiente ejemplo de una política basada en recursos no pública restringe sourceVPC al usar el operador StringEquals.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "dynamodb:*",
```

```
      "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/
MusicCollection",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": [
            "vpc-91237329"
          ]
        }
      }
    }
  ]
}
```

Operaciones de la API de DynamoDB admitidas por las políticas basadas en recursos

Este tema muestra las operaciones de la API admitidas por políticas basadas en recursos. Sin embargo, para acceso entre cuentas, puede usar solo un determinado conjunto de API de DynamoDB a través de políticas basadas en recursos. No puede asociar políticas basadas en recursos a tipos de recursos, como las copias de seguridad y las importaciones. Las acciones de IAM, que se corresponden con las API que funcionan en estos tipos de recursos, están excluidas de las acciones de IAM admitidas en las políticas basadas en recursos. Como los administradores de tablas configuran los ajustes internos de las tablas en la misma cuenta, las API, tales como UpdateTimetoLive y DisableKinesisStreamingDestination, no admiten el acceso entre cuentas mediante las políticas basadas en recursos.

Las API de plano de datos y plano de control de DynamoDB que admiten el acceso entre cuentas también admiten la sobrecarga de nombres de tabla, lo que permite especificar el ARN de la tabla en lugar del nombre de la tabla. Puede especificar el ARN de la tabla en el parámetro `TableName` de estas API. Sin embargo, no todas estas API admiten el acceso entre cuentas.

Temas

- Operaciones de la API del plano de datos
- Operaciones de API de PartiQL
- Operaciones de la API del plano de control
- Operaciones de API de tablas globales de la versión 2019.11.21 (actual)
- Operaciones de API de tablas globales de la versión 2017.11.29 (heredada)

- Operaciones de la API en etiquetas
- Operaciones de API de copia de seguridad y restauración
- Operaciones de API de copia de seguridad o restauración continuas (PITR)
- Operaciones de la API de Contributor Insights
- Operaciones de la API de exportación
- Operaciones de la API de importación
- Operaciones de API de Amazon Kinesis Data Streams
- Operaciones de API de políticas basadas en recursos
- Operaciones de API de tiempo de vida
- Otras operaciones de la API
- Operaciones de API de DynamoDB Streams

Operaciones de la API del plano de datos

En la siguiente tabla se muestra la compatibilidad en el nivel de la API proporcionada por las operaciones de la API de plano de datos para las políticas basadas en recursos y el acceso entre cuentas.

Plano de datos: API de tablas e índices	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
DeleteItem	Sí	Sí
GetItem	Sí	Sí
PutItem	Sí	Sí
Query	Sí	Sí
Examen	Sí	Sí
UpdateItem	Sí	Sí
TransactGetItems	Sí	Sí
TransactWriteItems	Sí	Sí

Plano de datos: API de tablas e índices	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
BatchGetItem	Sí	Sí
BatchWriteItem	Sí	Sí

Operaciones de API de PartiQL

En la siguiente tabla se muestra la compatibilidad en el nivel de la API proporcionada por las operaciones de la API de PartiQL para las políticas basadas en recursos y el acceso entre cuentas.

API de PartiQL	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
BatchExecuteStatement	Sí	No
ExecuteStatement	Sí	No
ExecuteTransaction	Sí	No

Operaciones de la API del plano de control

En la siguiente tabla se muestra la compatibilidad en el nivel de la API proporcionada por las operaciones de la API de plano de control para las políticas basadas en recursos y el acceso entre cuentas.

Plano de control: API de tablas	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
CreateTable	No	No
DeleteTable	Sí	Sí
DescribeTable	Sí	Sí
UpdateTable	Sí	Sí

Operaciones de API de tablas globales de la versión 2019.11.21 (actual)

En la siguiente tabla se muestra la compatibilidad en el nivel de la API proporcionada por las operaciones de la API de tablas globales de la versión 2019.11.21 (actual) para las políticas basadas en recursos y el acceso entre cuentas.

API de tablas globales de la versión 2019.11.21 (actual)	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
DescribeTableReplicaAutoScaling	Sí	No
UpdateTableReplicaAutoScaling	Sí	No

Operaciones de API de tablas globales de la versión 2017.11.29 (heredada)

En la siguiente tabla se muestra la compatibilidad en el nivel de la API proporcionada por las operaciones de la API de tablas globales de la versión 2017.11.29 (heredada) para las políticas basadas en recursos y el acceso entre cuentas.

API de tablas globales de la versión 2017.11.29 (heredada)	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
CreateGlobalTable	No	No
DescribeGlobalTable	No	No
DescribeGlobalTableSettings	No	No
ListGlobalTables	No	No
UpdateGlobalTable	No	No
UpdateGlobalTableSettings	No	No

Operaciones de la API en etiquetas

En la siguiente tabla se muestra la compatibilidad en el nivel de la API proporcionada por las operaciones de la API relacionadas con etiquetas para las políticas basadas en recursos y el acceso entre cuentas.

API de etiquetas	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
ListTagsOfResource	Sí	Sí
TagResource	Sí	Sí
UntagResource	Sí	Sí

Operaciones de API de copia de seguridad y restauración

En la siguiente tabla se muestra la compatibilidad en el nivel de la API proporcionada por las operaciones de la API relacionadas con copia de seguridad y restauración para las políticas basadas en recursos y el acceso entre cuentas.

API de copia de seguridad y restauración	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
CreateBackup	Sí	No
DescribeBackup	No	No
DeleteBackup	No	No
RestoreTableFromBackup	No	No

Operaciones de API de copia de seguridad o restauración continuas (PITR)

En la siguiente tabla se muestra la compatibilidad en el nivel de la API proporcionada por las operaciones de la API relacionadas con copia de seguridad o restauración continuas (PITR) para las políticas basadas en recursos y el acceso entre cuentas.

API de copia de seguridad o restauración continuas (PITR)	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
DescribeContinuousBackups	Sí	No
RestoreTableToPointInTime	Sí	No
UpdateContinuousBackups	Sí	No

Operaciones de la API de Contributor Insights

En la siguiente tabla se muestra la compatibilidad en el nivel de la API proporcionada por las operaciones de la API relacionadas con copia de seguridad o restauración continuas (PITR) para las políticas basadas en recursos y el acceso entre cuentas.

API de Contributor Insights	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
DescribeContributorInsights	Sí	No
ListContributorInsights	No	No
UpdateContributorInsights	Sí	No

Operaciones de la API de exportación

En la siguiente tabla se muestra la compatibilidad en el nivel de la API proporcionada por las operaciones de la API de exportación para las políticas basadas en recursos y el acceso entre cuentas.

API de exportación	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
DescribeExport	No	No
ExportTableToPointInTime	Sí	No

API de exportación	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
ListExports	No	No

Operaciones de la API de importación

En la siguiente tabla se muestra la compatibilidad en el nivel de la API proporcionada por las operaciones de la API de importación para las políticas basadas en recursos y el acceso entre cuentas.

API de importación	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
DescribeImport	No	No
ImportTable	No	No
ListImports	No	No

Operaciones de API de Amazon Kinesis Data Streams

En la siguiente tabla se muestra la compatibilidad en el nivel de la API proporcionada por las operaciones de la API de Kinesis Data Streams para las políticas basadas en recursos y el acceso entre cuentas.

API de Kinesis	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
DescribeKinesisStreamingDestination	Sí	No
Deshabilite el destino de transmisión de Kinesis	Sí	No
EnableKinesisStreamingDestination	Sí	No

API de Kinesis	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
UpdateKinesisStreamingDestination	Sí	No

Operaciones de API de políticas basadas en recursos

En la siguiente tabla se muestra la compatibilidad en el nivel de la API proporcionada por las operaciones de la API de la política basada en recursos para las políticas basadas en recursos y el acceso entre cuentas.

API de política basada en recursos	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
GetResourcePolicy	Sí	No
PutResourcePolicy	Sí	No
DeleteResourcePolicy	Sí	No

Operaciones de API de tiempo de vida

En la siguiente tabla se muestra la compatibilidad en el nivel de la API proporcionada por las operaciones de la API de tiempo de vida (TTL) para las políticas basadas en recursos y el acceso entre cuentas.

API de TTL	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
DescribeTimeToLive	Sí	No
UpdateTimeToLive	Sí	No

Otras operaciones de la API

En la siguiente tabla se muestra la compatibilidad en el nivel de la API proporcionada por otras diversas operaciones de la API para las políticas basadas en recursos y el acceso entre cuentas.

Otras API	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
DescribeLimits	No	No
DescribeEndpoints	No	No
ListBackups	No	No
ListTables	No	No

Operaciones de API de DynamoDB Streams

En la siguiente tabla se muestra la compatibilidad en el nivel de la API de las API de DynamoDB Streams para las políticas basadas en recursos y el acceso entre cuentas.

API de DynamoDB Streams	Compatibilidad con las políticas basadas en recursos	Compatibilidad entre cuentas
DescribeStream	Sí	Sí
GetRecords	Sí	Sí
GetShardIterator	Sí	Sí
ListStreams	No	No

Autorización con políticas basadas en identidad de IAM y políticas basadas en recursos de DynamoDB

Las políticas basadas en identidad están asociadas a una identidad, como los usuarios, los grupos de usuarios y roles de IAM. Estos son documentos de políticas de IAM que controlan las acciones

que puede realizar una identidad, en qué recursos y en qué condiciones. Las políticas basadas en identidad pueden ser administradas o insertadas.

Las políticas basadas en recursos son documentos de políticas JSON que se asocian a un recurso, como una tabla de DynamoDB. Estas políticas conceden a la entidad principal especificada permiso para ejecutar acciones concretas en el recurso y definen en qué condiciones son aplicables. Por ejemplo, la política basada en recursos de una tabla de DynamoDB también incluye el índice asociado a la tabla. Las políticas basadas en recursos son políticas insertadas. No existen políticas basadas en recursos que sean administradas.

Para obtener más información acerca de estas políticas, consulte Políticas basadas en identidad y políticas basadas en recursos en la Guía del usuario de IAM.

Si la entidad principal de IAM es de la misma cuenta que el propietario del recurso, la política basada en recursos es suficiente para especificar los permisos de acceso al recurso. También podrá optar por una política basada en identidades de IAM junto con una política basada en recursos. Para obtener acceso entre cuentas, debe permitir el acceso de forma explícita en las políticas de identidad y recursos, tal como se especifica en Acceso entre cuentas con políticas basadas en recursos en DynamoDB. Cuando utilice ambos tipos de políticas, una política se evaluará tal como se describe en Cómo determinar si una solicitud se permite o se deniega en una cuenta.

Ejemplos de política basada en recursos de DynamoDB

Al especificar un ARN en el campo `Resource` de una política basada en recursos, la política solo entra en vigor si el ARN especificado coincide con el ARN del recurso de DynamoDB al que está asociado.

> **Note**
>
> Recuerde reemplazar el texto en *`cursiva`* por la información específica del recurso.

Política basada en recursos para una tabla

La siguiente política basada en recursos asociada a una tabla de DynamoDB denominada *`MusicCollection`* otorga a los usuarios de IAM *`John`* y *`Jane`* permiso para realizar las acciones GetItem y BatchGetItem en el recurso *`MusicCollection`*.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
        "Effect": "Allow",
        "Principal": {
          "AWS": [
            "arn:aws:iam::111122223333:user/username",
            "arn:aws:iam::111122223333:user/Jane"
          ]
        },
        "Action": [
          "dynamodb:GetItem",
          "dynamodb:BatchGetItem"
        ],
        "Resource": [
          "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection"
        ]
    }
  ]
}
```

Política basada en recursos para una secuencia

La siguiente política basada en recursos asociada a una secuencia de DynamoDB denominada `2024-02-12T18:57:26.492` otorga a los usuarios de IAM *John* y *Jane* permiso para realizar las acciones de la API GetRecords, GetShardIterator y DescribeStream en el recurso `2024-02-12T18:57:26.492`.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
```

```
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::111122223333:user/username",
          "arn:aws:iam::111122223333:user/Jane"
        ]
      },
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords",
        "dynamodb:GetShardIterator"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
stream/2024-02-12T18:57:26.492"
      ]
    }
  ]
}
```

Política de acceso basada en recursos para realizar todas las acciones en recursos específicos

Para que un usuario pueda realizar todas las acciones en una tabla y en todos los índices asociados a ella, puede utilizar un comodín (*) para representar las acciones y los recursos asociados a la tabla. El uso de un carácter comodín para los recursos permitirá al usuario acceder a la tabla de DynamoDB y a todos los índices asociados, incluidos los que aún no se hayan creado. Por ejemplo, la siguiente política permitirá al usuario *John* realizar cualquier acción en la tabla *MusicCollection* y en todos sus índices, incluidos los índices que se creen en el futuro.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
```

```
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/role-name"
      },
      "Action": "dynamodb:*",
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/
index/index-name"
      ]
    }
  ]
}
```

Políticas basadas en recursos para el acceso entre cuentas

Puede especificar permisos para una identidad de IAM entre cuentas para acceder a los recursos de DynamoDB. Por ejemplo, es posible que necesite un usuario de una cuenta de confianza que pueda leer el contenido de su tabla, con la condición de que solo acceda a elementos y atributos específicos de esos elementos. En la siguiente política se permite al usuario *John* con un ID de Cuenta de AWS *111111111111* de confianza acceder a una tabla de la cuenta *123456789012* mediante la API GetItem. La política garantiza que el usuario solo pueda acceder a los elementos con la clave principal *Jane* y que solo pueda recuperar los atributos `Artist` y `SongTitle`, pero no otros atributos.

> **Important**
>
> Si no especifica la condición `SPECIFIC_ATTRIBUTES`, verá todos los atributos de los elementos devueltos.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "CrossAccountTablePolicy",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111111111111:user/John"
            },
            "Action": "dynamodb:GetItem",
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": "Jane",
                    "dynamodb:Attributes": [
                        "Artist",
                        "SongTitle"
                    ]
                },
                "StringEquals": {
                    "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
                }
            }
        }
    ]
}
```

Además de la anterior política basada en recursos, la política basada en identidad asociada al usuario *John* también debe permitir la acción de la API `GetItem` para que funcione el acceso entre cuentas. A continuación, se muestra un ejemplo de política basada en identidad que debe asociar al usuario *John*.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CrossAccountIdentityBasedPolicy",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem"
            ],
```

```
            "Resource": [
                "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": "Jane",
                    "dynamodb:Attributes": [
                        "Artist",
                        "SongTitle"
                    ]
                },
                "StringEquals": {
                    "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
                }
            }
        }
    ]
}
```

El usuario John puede hacer una solicitud `GetItem` especificando el ARN en el parámetro `table-name` para acceder a la tabla *`MusicCollection`* de la cuenta *`123456789012`*.

```
aws dynamodb get-item \
    --table-name arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
    --key '{"Artist": {"S": "Jane"}' \
    --projection-expression 'Artist, SongTitle' \
    --return-consumed-capacity TOTAL
```

Política basada en recursos con condiciones de dirección IP

Puede aplicar una condición para restringir las direcciones IP de origen, las nubes privadas virtuales (VPC) y los puntos de conexión de VPC (VPCE). Puede especificar los permisos en función de las direcciones de origen de la solicitud de origen. Por ejemplo, es posible que desee permitir que un usuario acceda a los recursos de DynamoDB solo si se accede a ellos desde una IP específica, como un punto de conexión de VPN corporativo. Especifique estas direcciones IP en la instrucción `Condition`.

En el siguiente ejemplo, el usuario *`John`* puede acceder a cualquier recurso de DynamoDB cuando las IP de origen son `54.240.143.0/24` y `2001:DB8:1234:5678::/64`.

JSON

```
{
  "Id":"PolicyId2",
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AllowIPmix",
      "Effect":"Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:user/username"
      },
      "Action":"dynamodb:*",
      "Resource":"*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": [
            "54.240.143.0/24",
            "2001:DB8:1234:5678::/64"
          ]
        }
      }
    }
  ]
}
```

También puede denegar todo acceso a los recursos de DynamoDB, excepto cuando el origen sea un punto de conexión de VPC específico, por ejemplo, *vpce-1a2b3c4d*.

> **Important**
>
> Cuando utiliza DAX con tablas de DynamoDB que tienen políticas de recursos basadas en IP en entornos que solo utilizan IPv6, debe configurar reglas de acceso adicionales. Si la política de recursos restringe el acceso al espacio de direcciones IPv4 `0.0.0.0/0` en las tablas, debe permitir el acceso para el rol de IAM asociado al clúster de DAX. Agregue una condición `ArnNotEquals` a la política para garantizar que DAX mantenga el acceso a las tablas de DynamoDB. Para obtener más información, consulte DAX e IPv6.

JSON

```
{
  "Id":"PolicyId",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AccessToSpecificVPCEOnly",
      "Principal": "*",
      "Action": "dynamodb:*",
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "StringNotEquals":{
          "aws:sourceVpce":"vpce-1a2b3c4d"
        }
      }
    }
  ]
}
```

Política basada en recursos que utiliza un rol de IAM

También puede especificar un rol de servicio de IAM en la política basada en recursos. Las entidades de IAM que asumen este rol están limitadas por las acciones permitidas especificadas para el rol y por el conjunto específico de recursos de la política basada en recursos.

En el siguiente ejemplo, se permite que una entidad de IAM realice todas las acciones de DynamoDB en los recursos *MusicCollection* y *MusicCollection* de DynamoDB.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "1111",
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::111122223333:role/role-name" },
      "Action": "dynamodb:*",
```

```
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection",
        "arn:aws:dynamodb:us-east-1:123456789012:table/MusicCollection/*"
      ]
    }
  ]
}
```

Consideraciones sobre la política basada en recursos de DynamoDB

Al definir políticas basadas en recursos para los recursos de DynamoDB, se aplican las siguientes consideraciones:

Consideraciones generales

- El tamaño máximo admitido para un documento de política basado en recursos es de 20 KB. DynamoDB cuenta los espacios en blanco al calcular el tamaño de una política según esta limitación.
- Las actualizaciones posteriores de una política para un recurso determinado se bloquean durante 15 segundos después de actualizar correctamente la política para el mismo recurso.
- Actualmente, solo puede asociar una política basada en recursos a las secuencias existentes. No puede asociar una política a una secuencia mientras la crea.

Consideraciones sobre la tabla global

- Las políticas basadas en recursos no se admiten en las réplicas de tablas globales de la versión 2017.11.29 (heredada).
- Dentro de una política basada en recursos, si se deniega la acción de un rol vinculado a un servicio (SLR) de DynamoDB para replicar los datos para una tabla global, se producirá un error al añadir o eliminar una réplica.
- El recurso AWS::DynamoDB::GlobalTable no permite crear una réplica ni añadir una política basada en recursos a dicha réplica en la misma actualización de pila en regiones distintas a la región en la que se implemente la actualización de la pila.

Consideraciones sobre el acceso entre cuentas

- El acceso entre cuentas mediante políticas basadas en recursos no admite tablas cifradas con claves administradas de AWS, ya que no se puede otorgar acceso entre cuentas a la política administrada por AWS de KMS.

CloudFormation Consideraciones sobre

- Las políticas basadas en recursos no admiten la detección de desviaciones. Si actualiza una política basada en recursos fuera de la plantilla de la pila de AWS CloudFormation, tendrá que actualizar la pila de CloudFormation con los cambios.
- Las políticas basadas en recursos no admiten cambios fuera de banda. Si agrega, actualiza o elimina una política fuera de la plantilla de CloudFormation, el cambio no se sobrescribirá si no hay cambios en la política en la plantilla.

  Por ejemplo, supongamos que la plantilla contiene una política basada en recursos que actualiza más tarde fuera de la plantilla. Si no realiza cambios en la política de la plantilla, la política actualizada en DynamoDB no se sincronizará con la política de la plantilla.

  Por el contrario, supongamos que la plantilla no contiene una política basada en recursos y se añade una política fuera de la plantilla. Esta política no se eliminará de DynamoDB mientras no se añada a la plantilla. Al añadir una política a la plantilla y actualizar la pila, la política existente en DynamoDB se actualizará para que coincida con la definida en la plantilla.

Prácticas recomendadas de las políticas basadas en recursos de DynamoDB

En este tema se describen las prácticas recomendadas para definir los permisos de acceso para los recursos de DynamoDB y las acciones permitidas en dichos recursos.

Simplificación del control de acceso a los recursos de DynamoDB

Si las entidades principales de AWS Identity and Access Management que necesitan acceder a un recurso de DynamoDB forman parte de la misma Cuenta de AWS que el propietario del recurso, no se requiere ninguna política de IAM basada en identidad para cada entidad principal. Bastará con asociar una política basada en recursos a los recursos pertinentes. Este tipo de configuración simplifica el control de acceso.

Protección de los recursos de DynamoDB con políticas basadas en recursos

Para todas las tablas y secuencias de DynamoDB, cree políticas basadas en recursos para aplicar el control de acceso a esos recursos. Las políticas basadas en recursos le permiten centralizar los permisos en el nivel de recursos, simplificar el control de acceso a las tablas, índices y secuencias de DynamoDB y reducir los gastos de administración. Si no se especifica ninguna política basada en recursos para una tabla o secuencia, se denegará implícitamente el acceso a la tabla o secuencia, a menos que las políticas basadas en identidad asociadas a las entidades principales de IAM permitan el acceso.

Aplicación de permisos de privilegios mínimos

Cuando establezca permisos con políticas basadas en recursos para los recursos de DynamoDB, conceda solo los permisos necesarios para llevar a cabo una acción. Para ello, debe definir las acciones que se pueden llevar a cabo en determinados recursos en condiciones específicas, también conocidos como permisos de privilegios mínimos. Puede empezar con permisos amplios mientras va conociendo los permisos que se necesitan para su carga de trabajo o caso de uso. A medida que su caso de uso vaya madurando, puede ir reduciendo los permisos que concede para alcanzar el objetivo de privilegio mínimo.

Análisis de la actividad de acceso entre cuentas para generar políticas de privilegios mínimos

IAM Access Analyzer informa del acceso entre cuentas a entidades externas específicas en las políticas basadas en recursos y proporciona visibilidad a la hora de refinar los permisos y cumplir con la política de privilegios mínimos. Para obtener más información acerca de la generación de políticas, consulte Generación de políticas de IAM Access Analyzer.

Uso de IAM Access Analyzer para generar políticas de privilegios mínimos

Para conceder solo los permisos necesarios para llevar a cabo una tarea, puede generar políticas que se basen en la actividad de acceso que haya iniciado sesión en AWS CloudTrail. IAM Access Analyzer analiza los servicios y acciones que utilizan sus políticas.

Uso del control de acceso basado en atributos con DynamoDB

El control de acceso basado en atributos (ABAC) es una estrategia de autorización que define los permisos de acceso en función de las condiciones de etiqueta de las políticas basadas en la

identidad u otras políticas de AWS, como las políticas basadas en recursos y las políticas de IAM de la organización. Puede asociar etiquetas a las tablas de DynamoDB, que después se evalúan en función de las condiciones basadas en etiquetas. Los índices asociados a una tabla heredan las etiquetas que agregue a la tabla. Puede agregar un máximo de 50 etiquetas para cada tabla de DynamoDB. El tamaño máximo admitido para todas las etiquetas de una tabla es de 10 KB. Para obtener más información sobre el etiquetado de recursos de DynamoDB y las restricciones de etiquetado, consulte Etiquetado de recursos en DynamoDB y the section called “Restricciones de etiquetado en DynamoDB”.

Para obtener más información sobre cómo utilizar etiquetas para controlar el acceso a los recursos de AWS, consulte los siguientes temas en la Guía del usuario de IAM:

- Qué es ABAC para AWS
- Control de acceso a los recursos de AWS mediante etiquetas

Con ABAC, puede aplicar distintos niveles de acceso para que los equipos y aplicaciones realicen acciones en las tablas de DynamoDB con menos políticas. Puede especificar una etiqueta en el elemento de condición de una política de IAM para controlar el acceso a las tablas o los índices de DynamoDB. Estas condiciones determinan el nivel de acceso que una entidad principal, un usuario o un rol de IAM tiene a las tablas y los índices de DynamoDB. Cuando una entidad principal de IAM realiza una solicitud de acceso a DynamoDB, las etiquetas del recurso y de la identidad se evalúan con respecto a las condiciones de etiqueta de la política de IAM. Posteriormente, la política entra en vigor solo si se cumplen las condiciones de etiqueta. De este modo, podrá crear una política de IAM que indique de forma efectiva una de las siguientes acciones:

- Permitir al usuario administrar solo los recursos que tengan una etiqueta con una clave *X* y un valor *Y*.
- Denegar el acceso a todos los usuarios a los recursos etiquetados con una clave *X*.

Por ejemplo, puede crear una política que permita a los usuarios actualizar una tabla solo si tiene el par clave-valor de etiqueta: `"environment": "staging"`. Puede utilizar la clave de condición aws:ResourceTag para permitir o denegar el acceso a una tabla en función de las etiquetas que estén asociadas a dicha tabla.

Puede incluir condiciones basadas en atributos al crear la política o, posteriormente, con la Consola de administración de AWS, la API de AWS, la AWS Command Line Interface (AWS CLI), AWS SDK o AWS CloudFormation.

El siguiente ejemplo permite la acción UpdateItem en una tabla denominada `MusicTable` si incluye una clave de etiqueta con el nombre `environment` y el valor `production`.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:UpdateItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/MusicTable",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/environment": "production"
        }
      }
    }
  ]
}
```

Temas

- ¿Por qué debo utilizar ABAC?
- Claves de condición para implementar ABAC con DynamoDB
- Consideraciones para utilizar ABAC con DynamoDB
- Habilitación de ABAC en DynamoDB
- Uso de ABAC con tablas e índices de DynamoDB
- Ejemplos para utilizar ABAC con tablas e índices de DynamoDB
- Solución de errores de ABAC común en tablas e índices de DynamoDB

¿Por qué debo utilizar ABAC?

- Administración de políticas más sencilla: utilizará menos políticas porque no tendrá que crear políticas diferentes para definir el nivel de acceso de cada entidad principal de IAM.

- Control de acceso escalable: el control de acceso de escalado es más fácil con ABAC porque no tiene que actualizar las políticas cuando crea nuevos recursos de DynamoDB. Puede utilizar etiquetas para autorizar el acceso a entidades principales de IAM que contengan etiquetas que coincidan con las etiquetas del recurso. Puede incorporar nuevas entidades principales de IAM o recursos de DynamoDB y aplicar las etiquetas adecuadas para conceder de forma automática los permisos necesarios sin tener que realizar ningún cambio de política.
- Administración de permisos detallada: es una práctica recomendada conceder el privilegio mínimo cuando se crean políticas. Con ABAC, puede crear etiquetas para la entidad principal de IAM y utilizarlas a fin de conceder acceso a acciones y recursos específicos que coincidan con las etiquetas de la entidad principal de IAM.
- Alineación con el directorio corporativo: puede asignar etiquetas con atributos de empleado existentes en el directorio corporativo para alinear las políticas de control de acceso con la estructura organizativa.

Claves de condición para implementar ABAC con DynamoDB

Puede utilizar las siguientes claves de condición en las políticas de AWS para controlar el nivel de acceso a las tablas y los índices de DynamoDB:

- aws:ResourceTag/tag-key: controla el acceso en función de si el par clave-valor de la etiqueta en una tabla o un índice de DynamoDB coincide o no con la clave y el valor de la etiqueta de una política. Esta clave de condición es relevante para todas las API que operan en una tabla o un índice existente.

  Las condiciones `dynamodb:ResourceTag` se evalúan como si no hubiera asociado ninguna etiqueta a un recurso.
- aws:RequestTag/tag-key: permite comparar el par clave-valor de etiqueta que se ha pasado en la solicitud con el par de etiqueta que se especifica en la política. Esta clave de condición es relevante para las API que contienen etiquetas como parte de la carga útil de la solicitud. Estas API incluyen CreateTable y TagResource.
- aws:TagKeys: compara las claves de etiqueta de una solicitud con las claves que especifique en la política. Esta clave de condición es relevante para las API que contienen etiquetas como parte de la carga útil de la solicitud. Estas API incluyen `CreateTable`, `TagResource` y `UntagResource`.

Consideraciones para utilizar ABAC con DynamoDB

Cuando utilice ABAC con tablas o índices de DynamoDB, se aplicarán las siguientes consideraciones:

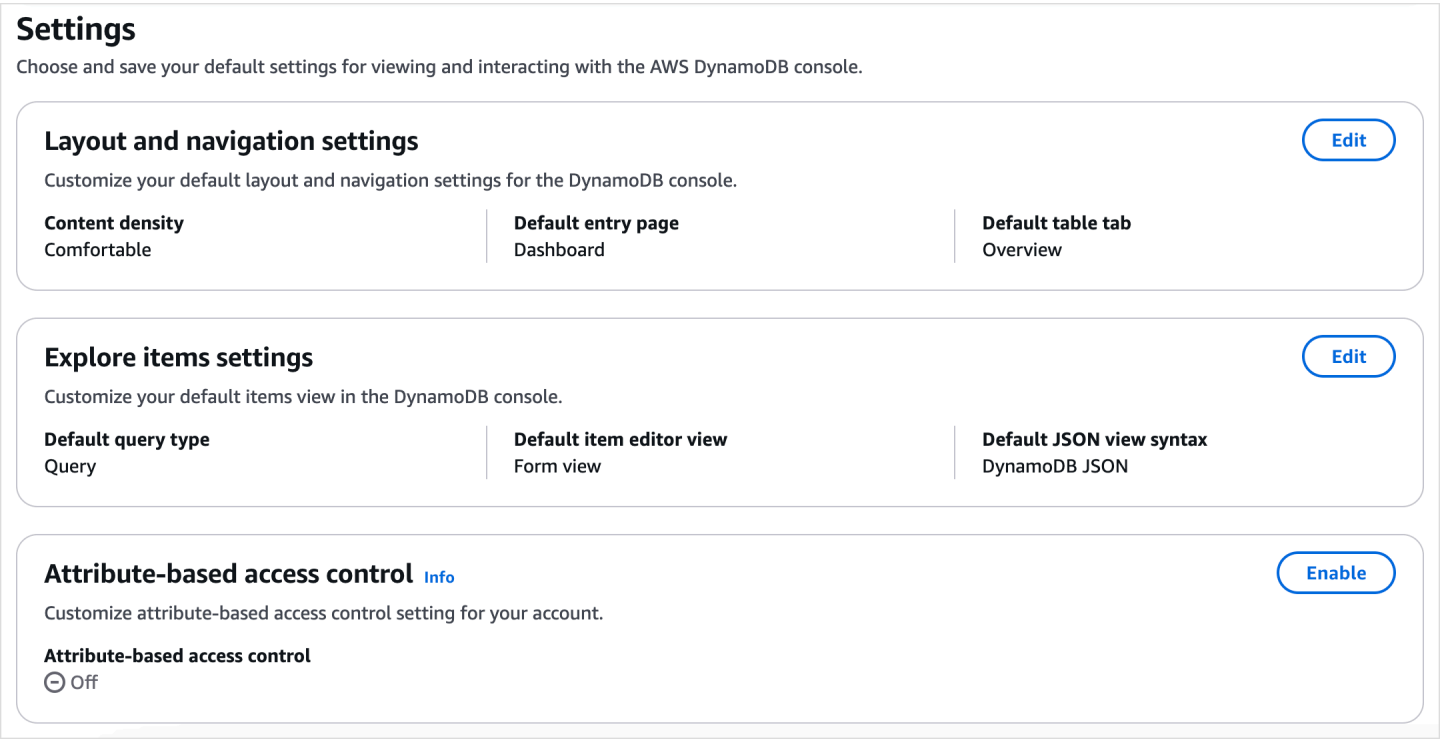
- El etiquetado y ABAC no son compatibles con DynamoDB Streams.
- El etiquetado y ABAC no son compatibles con las copias de seguridad de DynamoDB. Para utilizar ABAC con las copias de seguridad, le recomendamos que utilice AWS Backup.
- Las etiquetas no se conservan en las tablas restauradas. Debe agregar etiquetas a las tablas restauradas para poder utilizar las condiciones basadas en etiquetas en las políticas.

Habilitación de ABAC en DynamoDB

En la mayoría de las Cuentas de AWS, ABAC está activado de forma predeterminada. Mediante la consola de DynamoDB, puede confirmar si ABAC se ha habilitado para la cuenta. Para ello, asegúrese de abrir la consola de DynamoDB con un rol que tenga el permiso dynamodb:GetAbacStatus. A continuación, abra la página Configuración de la consola de DynamoDB.

Si no ve la tarjeta Control de acceso basado en atributos o si esta muestra el estado Activado, significa que ABAC se ha habilitado para la cuenta. No obstante, si ve la tarjeta Control de acceso basado en atributos con el estado Desactivado, como se muestra en la siguiente imagen, ABAC no se ha habilitado para la cuenta.

Control de acceso basado en atributos: no habilitado



ABAC no se ha habilitado para las Cuentas de AWS cuyas condiciones basadas en etiquetas especificadas en las políticas basadas en identidad u otras políticas aún deban auditarse. Si ABAC no se ha habilitado para la cuenta, las condiciones basadas en etiquetas de las políticas destinadas a actuar en tablas o índices de DynamoDB se evalúan como si no hubiera etiquetas presentes para los recursos o las solicitudes de la API. Cuando ABAC se ha habilitado para la cuenta, las condiciones basadas en etiquetas de las políticas de la cuenta se evalúan teniendo en cuenta las etiquetas asociadas a las tablas o a las solicitudes de la API.

Para habilitar ABAC en la cuenta, le recomendamos que primero audite las políticas tal y como se describe en la sección Auditoría de políticas. A continuación, incluya los permisos necesarios para ABAC en la política de IAM. Por último, realice los pasos descritos en Habilitación de ABAC en la consola para habilitar ABAC en la cuenta en la región actual. Después de habilitar ABAC, podrá desactivarlo en los siete días naturales posteriores a la activación.

Temas

- Auditoría de las políticas antes de habilitar ABAC
- Permisos de IAM necesarios para habilitar ABAC
- Habilitación de ABAC en la consola

Auditoría de las políticas antes de habilitar ABAC

Antes de activar ABAC en la cuenta, audite las políticas para confirmar que las condiciones basadas en etiquetas que puedan existir en las políticas de la cuenta están configuradas según lo previsto. La auditoría de las políticas lo ayudará a evitar sorpresas derivadas de los cambios de autorización en los flujos de trabajo de DynamoDB una vez habilitado ABAC. Para ver ejemplos de utilización de condiciones basadas en atributos con etiquetas y el comportamiento antes y después de la implementación de ABAC, consulte Ejemplos para utilizar ABAC con tablas e índices de DynamoDB.

Permisos de IAM necesarios para habilitar ABAC

Necesita el permiso `dynamodb:UpdateAbacStatus` para habilitar ABAC en la cuenta en la región actual. Para confirmar si ABAC se ha habilitado en la cuenta, también debe tener el permiso `dynamodb:GetAbacStatus`. Con este permiso, puede ver el estado de ABAC de una cuenta en cualquier región. Necesitará estos permisos además del permiso necesario para acceder a la consola de DynamoDB.

La siguiente política de IAM concede el permiso para habilitar ABAC y ver el estado para una cuenta en la región actual.

```
{
"version": "2012-10-17",&TCX5-2025-waiver;
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateAbacStatus",
                "dynamodb:GetAbacStatus"
             ],
            "Resource": "*"
        }
    ]
}
```

Habilitación de ABAC en la consola

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación superior, elija la región para la que desea habilitar ABAC.
3. En el panel de navegación izquierdo, elija Configuración.

4. En la página Configuración, haga lo siguiente:

   a. En la tarjeta Control de acceso basado en atributos, elija Habilitar.

   b. En el cuadro Confirmar la configuración del control de acceso basado en atributos, elija Habilitar para confirmar la elección.

      Esto habilita ABAC para la región actual y la tarjeta Control de acceso basado en atributos muestra el estado Activado.

      Si desea desactivar ABAC después de habilitarlo en la consola, puede hacerlo en los siete días naturales posteriores a la activación. Para desactivarlo, elija Deshabilitar en la tarjeta Control de acceso basado en atributos de la página Configuración.

      > **Note**
      >
      > La actualización del estado de ABAC es una operación asíncrona. Si las etiquetas de las políticas no se evalúan de inmediato, es posible que tenga que esperar algún tiempo hasta que la aplicación de los cambios sea coherente.

Uso de ABAC con tablas e índices de DynamoDB

En los pasos siguientes se muestra cómo configurar permisos con ABAC. En este escenario de ejemplo, agregará etiquetas a una tabla de DynamoDB y creará un rol de IAM con una política que incluya condiciones basadas en etiquetas. Después, probará los permisos permitidos en la tabla de DynamoDB mediante la coincidencia de las condiciones de etiqueta.

Temas

- Paso 1: Adición etiquetas a una tabla de DynamoDB
- Paso 2: Creación de un rol de IAM con una política que incluya condiciones basadas en etiquetas
- Paso 3: Prueba de los permisos permitidos

Paso 1: Adición etiquetas a una tabla de DynamoDB

Puede agregar etiquetas a tablas de DynamoDB nuevas o existentes mediante la Consola de administración de AWS, la API de AWS, la AWS Command Line Interface (AWS CLI), AWS SDK o

AWS CloudFormation. Por ejemplo, el siguiente comando de la CLI tag-resource agrega una etiqueta a una tabla denominada `MusicTable`.

```
aws dynamodb tag-resource –resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/
MusicTable –tags Key=environment,Value=staging
```

Paso 2: Creación de un rol de IAM con una política que incluya condiciones basadas en etiquetas

Cree una política de IAM mediante la clave de condición aws:ResourceTag/tag-key para comparar el par clave-valor de la etiqueta que se especifica en la política de IAM con el par clave-valor que está asociado a la tabla. La siguiente política de ejemplo permite a los usuarios colocar o actualizar elementos en tablas si estas contienen el par clave-valor de etiqueta: `"environment": "staging"`. Si una tabla no tiene el par clave-valor de etiqueta especificado, estas acciones se deniegan.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/environment": "staging"
                }
            }
        }
    ]
}
```

Paso 3: Prueba de los permisos permitidos

1. Asocie la política de IAM a un usuario o rol de prueba en la Cuenta de AWS. Asegúrese de que la entidad principal de IAM que utiliza ya no tiene acceso a la tabla de DynamoDB a través de una política diferente.
2. Asegúrese de que la tabla de DynamoDB contiene la clave de etiqueta `"environment"` con un valor de `"staging"`.
3. Realice las acciones `dynamodb:PutItem` y `dynamodb:UpdateItem` en la tabla etiquetada. Estas acciones deberían realizarse de forma correcta si el par clave-valor de etiqueta `"environment": "staging"` está presente.

   Si lleva a cabo estas acciones en una tabla que no tiene el par clave-valor de etiqueta `"environment": "staging"`, se producirá el error `AccessDeniedException` en la solicitud.

También puede revisar los otros casos de uso de ejemplo descritos en la siguiente sección para implementar ABAC y realizar más pruebas.

Ejemplos para utilizar ABAC con tablas e índices de DynamoDB

En los siguientes ejemplos se describen algunos casos de uso para implementar condiciones basadas en atributos mediante etiquetas.

Temas

- Ejemplo 1: Permitir una acción con aws:ResourceTag
- Ejemplo 2: Concesión de permiso a una acción mediante aws:RequestTag
- Ejemplo 3: Denegación de una acción mediante aws:TagKeys

Ejemplo 1: Permitir una acción con aws:ResourceTag

Con la clave de condición `aws:ResourceTag/tag-key`, puede comparar el par clave-valor de etiqueta que está especificado en una política de IAM con el par clave-valor que está asociado con una tabla de DynamoDB. Por ejemplo, puede permitir una acción específica, como PutItem, si las condiciones de etiqueta coinciden en una política de IAM y en una tabla. Para ello, siga estos pasos:

Using the AWS CLI

1. Crear una tabla. En el siguiente ejemplo se utiliza el comando create-table de la AWS CLI para crear una tabla denominada `myMusicTable`.

```
aws dynamodb create-table \
  --table-name myMusicTable \
  --attribute-definitions AttributeName=id,AttributeType=S \
  --key-schema AttributeName=id,KeyType=HASH \
  --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
  --region us-east-1
```

2. Agregue una etiqueta a esta tabla. El siguiente ejemplo de comando tag-resource de la AWS CLI agrega el par clave-valor de etiqueta `Title: ProductManager` a `myMusicTable`.

```
aws dynamodb tag-resource --region us-east-1 --resource-arn arn:aws:dynamodb:us-east-1:123456789012:table/myMusicTable --tags Key=Title,Value=ProductManager
```

3. Cree una política insertada y agréguela a un rol que tenga asociada la política administrada AmazonDynamoDBReadOnlyAccess de AWS, como se muestra en el siguiente ejemplo.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "dynamodb:PutItem",
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Title": "ProductManager"
        }
      }
    }
  ]
}
```

Esta política permite la `PutItem` en la tabla cuando la clave y el valor de etiqueta asociados a la tabla coinciden con las etiquetas especificadas en la política.

4. Asuma el rol con las políticas descritas en el paso 3.
5. Utilice el comando put-item de la AWS CLI para colocar un elemento en `myMusicTable`.

```
aws dynamodb put-item \
    --table-name myMusicTable --region us-east-1 \
    --item '{
        "id": {"S": "2023"},
        "title": {"S": "Happy Day"},
        "info": {"M": {
            "rating": {"N": "9"},
            "Artists": {"L": [{"S": "Acme Band"}, {"S": "No One You Know"}]},
            "release_date": {"S": "2023-07-21"}
        }}
    }'
```

6. Examine la tabla para comprobar si el elemento se ha agregado a ella.

```
aws dynamodb scan --table-name myMusicTable  --region us-east-1
```

Using the AWS SDK for Java 2.x

1. Crear una tabla. En el siguiente ejemplo se utiliza la API CreateTable para crear una tabla denominada `myMusicTable`.

```
DynamoDbClient dynamoDB = DynamoDbClient.builder().region(region).build();
CreateTableRequest createTableRequest = CreateTableRequest.builder()
    .attributeDefinitions(
        Arrays.asList(
            AttributeDefinition.builder()
            .attributeName("id")
            .attributeType(ScalarAttributeType.S)
            .build()
        )
    )
    .keySchema(
        Arrays.asList(
            KeySchemaElement.builder()
```

```
            .attributeName("id")
            .keyType(KeyType.HASH)
            .build()
        )
    )
    .provisionedThroughput(ProvisionedThroughput.builder()
        .readCapacityUnits(5L)
        .writeCapacityUnits(5L)
        .build()
    )
    .tableName("myMusicTable")
    .build();

CreateTableResponse createTableResponse =
 dynamoDB.createTable(createTableRequest);
String tableArn = createTableResponse.tableDescription().tableArn();
String tableName = createTableResponse.tableDescription().tableName();
```

2. Agregue una etiqueta a esta tabla. La API TagResource del siguiente ejemplo agrega el par clave-valor de etiqueta `Title: ProductManager` a `myMusicTable`.

```
TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
    .resourceArn(tableArn)
    .tags(
        Arrays.asList(
            Tag.builder()
            .key("Title")
            .value("ProductManager")
            .build()
        )
    )
    .build();
dynamoDB.tagResource(tagResourceRequest);
```

3. Cree una política insertada y agréguela a un rol que tenga asociada la política administrada AmazonDynamoDBReadOnlyAccess de AWS, como se muestra en el siguiente ejemplo.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": "dynamodb:PutItem",
      "Resource": "arn:aws:dynamodb:*:*:table/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Title": "ProductManager"
        }
      }
    }
  ]
}
```

Esta política permite la `PutItem` en la tabla cuando la clave y el valor de etiqueta asociados a la tabla coinciden con las etiquetas especificadas en la política.

4. Asuma el rol con las políticas descritas en el paso 3.
5. Utilice la API PutItem para colocar un elemento en `myMusicTable`.

```
HashMap<String, AttributeValue> info = new HashMap<>();
info.put("rating", AttributeValue.builder().s("9").build());
info.put("artists", AttributeValue.builder().ss(List.of("Acme Band","No One You
 Know").build());
info.put("release_date", AttributeValue.builder().s("2023-07-21").build());

HashMap<String, AttributeValue> itemValues = new HashMap<>();
itemValues.put("id", AttributeValue.builder().s("2023").build());
itemValues.put("title", AttributeValue.builder().s("Happy Day").build());
itemValues.put("info", AttributeValue.builder().m(info).build());


PutItemRequest putItemRequest = PutItemRequest.builder()
                .tableName(tableName)
                .item(itemValues)
                .build();
dynamoDB.putItem(putItemRequest);
```

6. Examine la tabla para comprobar si el elemento se ha agregado a ella.

```
ScanRequest scanRequest = ScanRequest.builder()
                .tableName(tableName)
                .build();
```

```
ScanResponse scanResponse = dynamoDB.scan(scanRequest);
```

Using the AWS SDK for Python (Boto3)

1. Crear una tabla. En el siguiente ejemplo se utiliza la API CreateTable para crear una tabla denominada `myMusicTable`.

```
create_table_response = ddb_client.create_table(
    AttributeDefinitions=[
        {
            'AttributeName': 'id',
            'AttributeType': 'S'
        },
    ],
    TableName='myMusicTable',
    KeySchema=[
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'
        },
    ],
        ProvisionedThroughput={
        'ReadCapacityUnits': 5,
        'WriteCapacityUnits': 5
    },
)

table_arn = create_table_response['TableDescription']['TableArn']
```

2. Agregue una etiqueta a esta tabla. La API TagResource del siguiente ejemplo agrega el par clave-valor de etiqueta `Title: ProductManager` a `myMusicTable`.

```
tag_resouce_response = ddb_client.tag_resource(
    ResourceArn=table_arn,
    Tags=[
        {
            'Key': 'Title',
            'Value': 'ProductManager'
        },
    ]
```

```
)
```

3. Cree una política insertada y agréguela a un rol que tenga asociada la política administrada AmazonDynamoDBReadOnlyAccess de AWS, como se muestra en el siguiente ejemplo.

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
           "Effect": "Allow",
           "Action": "dynamodb:PutItem",
           "Resource": "arn:aws:dynamodb:*:*:table/*",
           "Condition": {
               "StringEquals": {
               "aws:ResourceTag/Title": "ProductManager"
               }
           }
           }
       ]
       }
   ```

   Esta política permite la `PutItem` en la tabla cuando la clave y el valor de etiqueta asociados a la tabla coinciden con las etiquetas especificadas en la política.

4. Asuma el rol con las políticas descritas en el paso 3.

5. Utilice la API PutItem para colocar un elemento en `myMusicTable`.

   ```
   put_item_response = client.put_item(
       TableName = 'myMusicTable'
       Item = {
           'id': '2023',
           'title': 'Happy Day',
           'info': {
               'rating': '9',
               'artists': ['Acme Band','No One You Know'],
               'release_date': '2023-07-21'
           }
       }
   ```

```
)
```

6. Examine la tabla para comprobar si el elemento se ha agregado a ella.

```
scan_response = client.scan(
    TableName='myMusicTable'
)
```

Sin ABAC

Si ABAC no está habilitado para la Cuenta de AWS, las condiciones de etiquetado de la política de IAM y la tabla de DynamoDB no coinciden. En consecuencia, la acción `PutItem` devuelve `AccessDeniedException` debido al efecto de la política `AmazonDynamoDBReadOnlyAccess`.

```
An error occurred (AccessDeniedException) when calling the PutItem operation:
 User: arn:aws:sts::123456789012:assumed-role/DynamoDBReadOnlyAccess/Alice is
 not authorized to perform: dynamodb:PutItem on resource: arn:aws:dynamodb:us-
east-1:123456789012:table/myMusicTable because no identity-based policy allows the
 dynamodb:PutItem action.
```

Con ABAC

Si ABAC está habilitado para la Cuenta de AWS, la acción `put-item` se completa correctamente y agrega un nuevo elemento a la tabla. Esto se debe a que la política insertada en la tabla permite la acción `PutItem` si coinciden las condiciones de etiqueta de la política de IAM y de la tabla.

Ejemplo 2: Concesión de permiso a una acción mediante aws:RequestTag

Con la clave de condición aws:RequestTag/tag-key, puede comparar el par clave-valor de etiqueta que se pasa en la solicitud con el par de etiquetas especificado en la política de IAM. Por ejemplo, puede permitir una acción específica, como `CreateTable`, mediante `aws:RequestTag` si las condiciones de etiqueta no coinciden. Para ello, siga estos pasos:

Using the AWS CLI

1. Cree una política insertada y agréguela a un rol que tenga asociada la política administrada AmazonDynamoDBReadOnlyAccess de AWS, como se muestra en el siguiente ejemplo.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "John"
                }
            }
        }
    ]
}
```

2. Cree una tabla que contenga el par clave-valor de etiqueta de `"Owner": "John"`.

```
aws dynamodb create-table \
--attribute-definitions AttributeName=ID,AttributeType=S \
--key-schema AttributeName=ID,KeyType=HASH  \
--provisioned-throughput ReadCapacityUnits=1000,WriteCapacityUnits=500 \
--region us-east-1 \
--tags Key=Owner,Value=John \
--table-name myMusicTable
```

Using the AWS SDK for Python (Boto3)

1. Cree una política insertada y agréguela a un rol que tenga asociada la política administrada AmazonDynamoDBReadOnlyAccess de AWS, como se muestra en el siguiente ejemplo.

   JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "John"
                }
            }
        }
    ]
}
```

2. Cree una tabla que contenga el par clave-valor de etiqueta de `"Owner": "John"`.

```
ddb_client = boto3.client('dynamodb')

create_table_response = ddb_client.create_table(
    AttributeDefinitions=[
        {
            'AttributeName': 'id',
            'AttributeType': 'S'
        },
    ],
    TableName='myMusicTable',
    KeySchema=[
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'
        },
    ],
        ProvisionedThroughput={
        'ReadCapacityUnits': 1000,
        'WriteCapacityUnits': 500
    },
    Tags=[
        {
            'Key': 'Owner',
            'Value': 'John'
```

```
        },
    ],
)
```

Sin ABAC

Si ABAC no está habilitado para la Cuenta de AWS, las condiciones de etiquetado de la política insertada y la tabla de DynamoDB no coinciden. En consecuencia, se produce un error en la solicitud `CreateTable` y no se crea la tabla.

```
An error occurred (AccessDeniedException) when calling the CreateTable operation:
 User: arn:aws:sts::123456789012:assumed-role/Admin/John is not authorized to perform:
 dynamodb:CreateTable on resource: arn:aws:dynamodb:us-east-1:123456789012:table/
myMusicTable because no identity-based policy allows the dynamodb:CreateTable action.
```

Con ABAC

Si ABAC está habilitado para la Cuenta de AWS, la solicitud de creación de tabla se completa correctamente. Como el par clave-valor de etiqueta de `"Owner": "John"` está presente en la solicitud `CreateTable`, la política insertada permite al usuario `John` realizar la acción `CreateTable`.

Ejemplo 3: Denegación de una acción mediante aws:TagKeys

Con la clave de condición aws:TagKeys puede comparar las claves de etiqueta de una solicitud con las claves especificadas en la política de IAM. Por ejemplo, puede denegar una acción específica, como `CreateTable`, mediante `aws:TagKeys` si una clave de etiqueta específica no está presente en la solicitud. Para ello, siga estos pasos:

Using the AWS CLI

1. Agregue una política administrada por el cliente a un rol que tenga asociada la política administrada AmazonDynamoDBFullAccess de AWS, como se muestra en el siguiente ejemplo.

   JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "Null": {
                    "aws:TagKeys": "false"
                },
                "ForAllValues:StringNotEquals": {
                    "aws:TagKeys": "CostCenter"
                }
            }
        }
    ]
}
```

2. Asuma el rol al que se ha asociado la política y cree una tabla con la clave de etiqueta `Title`.

```
aws dynamodb create-table \
--attribute-definitions AttributeName=ID,AttributeType=S \
--key-schema AttributeName=ID,KeyType=HASH  \
--provisioned-throughput ReadCapacityUnits=1000,WriteCapacityUnits=500 \
--region us-east-1 \
--tags Key=Title,Value=ProductManager \
--table-name myMusicTable
```

Using the AWS SDK for Python (Boto3)

1. Agregue una política administrada por el cliente a un rol que tenga asociada la política administrada AmazonDynamoDBFullAccess de AWS, como se muestra en el siguiente ejemplo.

   JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:TagResource"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "Null": {
                    "aws:TagKeys": "false"
                },
                "ForAllValues:StringNotEquals": {
                    "aws:TagKeys": "CostCenter"
                }
            }
        }
    ]
}
```

2. Asuma el rol al que se ha asociado la política y cree una tabla con la clave de etiqueta `Title`.

```
ddb_client = boto3.client('dynamodb')

create_table_response = ddb_client.create_table(
    AttributeDefinitions=[
        {
            'AttributeName': 'id',
            'AttributeType': 'S'
        },
    ],
    TableName='myMusicTable',
    KeySchema=[
        {
            'AttributeName': 'id',
            'KeyType': 'HASH'
        },
    ],
        ProvisionedThroughput={
        'ReadCapacityUnits': 1000,
        'WriteCapacityUnits': 500
    },
```

```
    Tags=[
        {
            'Key': 'Title',
            'Value': 'ProductManager'
        },
    ],
)
```

Sin ABAC

Si ABAC no está habilitado para la Cuenta de AWS, DynamoDB no envía las claves de etiqueta del comando `create-table` a IAM. La condición `Null` garantiza que la condición se evalúe como `false` si no hay claves de etiqueta en la solicitud. Como la política `Deny` no coincide, el comando `create-table` se completa correctamente.

Con ABAC

Si ABAC está habilitado para la Cuenta de AWS, las claves de etiqueta pasadas en el comando `create-table` se pasan a IAM. La clave de etiqueta `Title` se evalúa con respecto a la clave de etiqueta basada en condiciones, `CostCenter`, presente en la política `Deny`. La clave de etiqueta `Title` no coincide con la clave de etiqueta presente en la política `Deny` debido al operador `StringNotEquals`. Por lo tanto, se produce un error en la acción `CreateTable` y la tabla no se crea. Al ejecutar el comando `create-table`, se devuelve `AccessDeniedException`.

```
An error occurred (AccessDeniedException) when calling the CreateTable operation:
 User: arn:aws:sts::123456789012:assumed-role/DynamoFullAccessRole/ProductManager is
 not authorized to perform: dynamodb:CreateTable on resource: arn:aws:dynamodb:us-
east-1:123456789012:table/myMusicTable with an explicit deny in an identity-based
 policy.
```

Solución de errores de ABAC común en tablas e índices de DynamoDB

En este tema se proporcionan consejos para la solución de problemas y errores comunes que puede encontrar al implementar ABAC en tablas o índices de DynamoDB.

Las claves de condición específicas del servicio en las políticas producen un error

Las claves de condición específicas del servicio no se consideran claves de condición válidas. Si ha utilizado este tipo de claves en las políticas, se producirá un error. Para solucionar este

problema, debe reemplazar las claves de condición específicas del servicio por una clave de condición adecuada para implementar ABAC en DynamoDB.

Por ejemplo, supongamos que ha utilizado la clave de condición `dynamodb:ResourceTag` en una política insertada que realiza la solicitud PutItem. Imagine que se produce el error `AccessDeniedException` en la solicitud. En el siguiente ejemplo se muestra la política insertada errónea con la clave de condición `dynamodb:ResourceTag`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "dynamodb:ResourceTag/Owner": "John"
                }
            }
        }
    ]
}
```

Para solucionar este problema, reemplace la clave de condición `dynamodb:ResourceTag` por `aws:ResourceTag`, como se muestra en el siguiente ejemplo.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:PutItem"
```

```
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "John"
                }
            }
        }
    ]
}
```

No se puede desactivar ABAC

Si se ha habilitado ABAC para la cuenta a través de Soporte, no podrá desactivarlo mediante la consola de DynamoDB. Para desactivarlo, póngase en contacto con Soporte.

Puede desactivar ABAC solo si se cumplen las siguientes condiciones:

- Ha utilizado la forma de autoservicio de activarlo a través de la consola de DynamoDB.
- Efectúa la desactivación en los siete días naturales posteriores a la activación.

Protección de datos en DynamoDB

Amazon DynamoDB proporciona una infraestructura de almacenamiento de larga duración diseñada para el almacenamiento de datos principales y críticos. Los datos se almacenan de forma redundante en varios dispositivos de diversas instalaciones dentro de una región de Amazon DynamoDB.

DynamoDB protege los datos del usuario almacenados en reposo y también los datos en tránsito entre los clientes locales y DynamoDB, y entre DynamoDB y otros recursos de AWS dentro de la misma región de AWS.

Temas

- Cifrado en reposo en DynamoDB
- Protección de las conexiones de DynamoDB mediante puntos de conexión de VPC y políticas de IAM"

Cifrado en reposo en DynamoDB

Todos los datos de usuario almacenados en Amazon DynamoDB están completamente cifrados en reposo. El cifrado en reposo de DynamoDB proporciona mayor seguridad, porque cifra todos sus datos en reposo mediante las claves de cifrado almacenadas en AWS Key Management Service (AWS KMS). Esta funcionalidad ayuda a reducir la carga y la complejidad operativas que conlleva la protección de información confidencial. Con el cifrado en reposo, puede crear aplicaciones sensibles a la seguridad que necesitan cumplimiento estricto de cifrado y requisitos normativos.

El cifrado en reposo de DynamoDB proporciona una capa adicional de seguridad de los datos, porque los protege en una tabla cifrada que incluye su clave principal, los índices secundarios locales y globales, las transmisiones, las tablas globales, las copias de seguridad y los clústeres de DynamoDB Accelerator (DAX) siempre que los datos se almacenen en un soporte duradero. Las políticas de la organización, las normativas industriales o gubernamentales y los requisitos de conformidad suelen requerir el uso del cifrado en reposo para aumentar la seguridad de los datos de las aplicaciones. Para obtener más información sobre el cifrado de aplicaciones de bases de datos, consulte AWS Database Encryption SDK.

El cifrado en reposo se integra con AWS KMS para administrar las claves de cifrado que se utilizan para cifrar las tablas. Para obtener más información sobre los tipos y estados de las claves, consulte los conceptos de AWS Key Management Service en la Guía para desarrolladores de AWS Key Management Service.

Al crear una tabla nueva, puede elegir uno de los siguientes tipos de AWS KMS key para cifrarla. Puede cambiar entre estos tipos de claves en cualquier momento.

- Clave propiedad de AWS – tipo de cifrado predeterminado. La clave es propiedad de DynamoDB (sin cargo adicional).
- Clave administrada de AWS: la clave se almacena en la cuenta y la administra AWS KMS (se aplican los cargos de AWS KMS).
- Customer managed key (Clave administrada por el cliente): la clave se almacena en la cuenta y usted la crea, posee y administra. Usted controla plenamente la clave KMS (se aplican los cargos de AWS KMS).

Para obtener más información sobre estos tipos de claves, consulte Customer keys and AWS keys.

> **Note**
>
> - Al crear un nuevo clúster DAX con el cifrado en reposo habilitado, se utilizará una Clave administrada de AWS para cifrar datos en reposo en el clúster.
> - Si la tabla tiene una clave de clasificación, algunas de las claves de ordenación que marcan los límites del rango se almacenan en texto no cifrado en los metadatos de la tabla.

Cuando accede a una tabla cifrada, DynamoDB descifra los datos de la tabla de forma transparente. No es necesario que cambie sus aplicaciones o código para utilizar o administrar tablas cifradas. DynamoDB continúa proporcionando la misma latencia en milisegundos de un solo dígito que cabe esperar de nosotros y todas las consultas de DynamoDB funcionan sin inconvenientes con los datos cifrados.

Puede especificar una clave de cifrado al crear una tabla o cambiar las claves de cifrado en una tabla existente con la Consola de administración de AWS, la AWS Command Line Interface (AWS CLI) o la API de Amazon DynamoDB. Para aprender a hacerlo, consulte Administración de tablas de cifrado en DynamoDB.

El cifrado en reposo mediante la Clave propiedad de AWS se ofrece sin cargo adicional. Sin embargo, se aplicarán cargos de AWS KMS por una Clave administrada de AWS y por una clave administrada por el cliente. Para obtener más información acerca de los precios, consulte Precios de AWS KMS.

El cifrado en reposo de DynamoDB está disponible en todas las regiones de AWSRegions, incluidas las regiones AWS China (Pekín) y AWS China (Ningxia), así como la región AWS GovCloud (EE. UU.). Para obtener más información, consulte Cifrado de DynamoDB en reposo: funcionamiento y Notas de uso del cifrado en reposo de DynamoDB.

Cifrado de DynamoDB en reposo: funcionamiento

El cifrado en reposo de Amazon DynamoDB cifra sus datos mediante cifrado estándar avanzado de 256 bits (AES-256), que ayuda a proteger sus datos del acceso no autorizado al almacenamiento subyacente.

El cifrado en reposo se integra con AWS Key Management Service (AWS KMS) para administrar las claves de cifrado que se utilizan para cifrar las tablas.

> **Note**
>
> En mayo de 2022, AWS KMS ha cambiado la programación de rotación para Claves administradas por AWS de cada tres años (aproximadamente 1095 días) hasta cada año (aproximadamente 365 días).
> Las nuevas Claves administradas por AWS rotan automáticamente un año después de su creación y, aproximadamente, cada año a partir de entonces.
> Las Claves administradas por AWS existentes rotan automáticamente un año después de su rotación más reciente y cada año a partir de entonces.

Claves propiedad de AWS

Las Claves propiedad de AWS no están almacenadas en su cuenta de AWS. Forman parte de una recopilación de claves KMS que AWS posee y administra para su uso en múltiples cuentas de AWS. Los servicios de AWS pueden usar las Claves propiedad de AWS para proteger los datos. Las Claves propiedad de AWS que usa DynamoDB se rotan cada año (aproximadamente 365 días).

No puede ver, administrar o usar las Claves propiedad de AWS, ni auditar su uso. Sin embargo, no es necesario que realice ninguna acción ni que cambie programas para proteger las claves que cifran sus datos.

No se le cobra ninguna tarifa mensual ni tarifa de uso de las Claves propiedad de AWS y no se incluyen en los límites de AWS KMS de su cuenta.

Claves administradas por AWS

Las Claves administradas por AWS son claves KMS de la cuenta que se crean, administran y utilizan en su nombre por un servicio de AWS integrado con AWS KMS. Puede ver las Claves administradas por AWS en su cuenta, ver sus políticas de claves y auditar su uso en los registros de AWS CloudTrail. Sin embargo, no puede administrar estas claves KMS ni modificar sus permisos.

El cifrado en reposo se integra automáticamente con AWS KMS para administrar las Claves administradas por AWS para DynamoDB (`aws/dynamodb`) que se utilizan para cifrar las tablas. Si no existe una Clave administrada de AWS al crear la tabla de DynamoDB cifrada, AWS KMS crea automáticamente una nueva clave. Esta clave se utilizará con las tablas cifradas que se creen en el futuro. AWS KMS combina hardware y software seguros de alta disponibilidad para ofrecer un sistema de administración de claves adaptado a la nube.

Para obtener más información acerca de la administración de permisos de las Clave administrada de AWS, consulte Autorización del uso de Clave administrada de AWS en la Guía para desarrolladores de AWS Key Management Service.

Claves administradas por el cliente

Las claves administradas por el cliente son claves KMS en su cuenta de AWS, que usted ha creado, posee y administra. Usted tiene un control total sobre estas claves KMS, incluyendo el establecimiento y mantenimiento de sus políticas de claves, políticas de IAM y concesiones, la habilitación y desactivación de las mismas, la rotación de su material criptográfico, la adición de etiquetas, la creación de alias que hacen referencia a ellas y la programación para su eliminación. Para obtener más información sobre la administración de permisos de una clave administrada por el cliente, consulte Claves administradas por el cliente.

Cuando se especifica una clave administrada por el cliente como clave de cifrado a nivel de tabla, la tabla de DynamoDB, los índices secundarios locales y globales y las transmisiones se cifran con la misma clave administrada por el cliente. Las copias de seguridad en diferido se cifran con la clave de cifrado de nivel de tabla que se especifica en el momento en que se crea la copia de seguridad. La actualización de la clave de cifrado de nivel de tabla no cambia la clave de cifrado asociada a las copias de seguridad en diferido existentes.

Configurar el estado de la clave administrada por el cliente como desactivado o programarlo para su eliminación impide que todos los usuarios y el servicio DynamoDB puedan cifrar o descifrar datos y realizar operaciones de lectura y escritura en la tabla. DynamoDB debe tener acceso a la clave de cifrado para asegurarse de que pueda seguir accediendo a la tabla y evitar la pérdida de datos.

Si desactiva la clave administrada por el cliente o la programa para que se elimine, el estado de la tabla se convierte en Inaccesible. Para asegurarse de que puede continuar trabajando con la tabla, debe darle a DynamoDB acceso a la clave de cifrado especificada en un plazo de siete días. Tan pronto como el servicio detecte que la clave de cifrado es inaccesible, DynamoDB le envía una notificación por correo electrónico para avisarle.

> **Note**
>
> - Si su clave administrada por el cliente permanece inaccesible para el servicio de DynamoDB durante más de siete días, la tabla se archiva y ya no podrá acceder a ella. DynamoDB crea una copia de seguridad en diferido de la tabla y se le emite una factura por ella. Puede utilizar esta copia de seguridad en diferido para restaurar los datos en una

nueva tabla. Para iniciar la restauración, la última clave administrada por el cliente en la tabla debe estar habilitada y DynamoDB debe tener acceso a ella.

- Si la clave administrada por el cliente que se utilizó para cifrar una réplica de tabla global es inaccesible, DynamoDB quitará esta réplica del grupo de replicación. La réplica no se eliminará y la replicación se detendrá desde y hacia esta región, 20 horas después de detectar que la clave administrada por el cliente es inaccesible.

Para obtener más información, consulte Habilitar claves y Eliminar claves.

Notas sobre el uso de las Claves administradas por AWS

Amazon DynamoDB no puede leer los datos de la tabla a menos que tenga acceso a la clave KMS almacenada en su cuenta de AWS KMS. DynamoDB utiliza el cifrado de envoltura y la jerarquía de claves para cifrar los datos. Sus clave de cifrado AWS KMS se utiliza para cifrar la clave raíz de esta jerarquía de claves. Para obtener más información, consulte Cifrado de sobre en la Guía para desarrolladores de AWS Key Management Service.

DynamoDB no llama a AWS KMS por cada operación de DynamoDB. La clave se actualiza una vez cada 5 minutos por intermediario con tráfico activo.

Asegúrese de haber configurado el SDK para que reutilice las conexiones. De lo contrario, experimentará latencias de DynamoDB al tener que restablecer nuevas entradas de caché de AWS KMS por cada operación de DynamoDB. Además, es posible que tenga que enfrentarse a costes de AWS KMS y CloudTrail más altos. Por ejemplo, para hacer esto usando el SDK Node.js, puede crear un nuevo agente HTTPS con `keepAlive` activado. Para obtener más información, consulte Configuración de keepAlive en Node.js en la Guía para desarrolladores de AWS SDK para JavaScript.

Notas de uso del cifrado en reposo de DynamoDB

Es importante tener en cuenta lo siguiente cuando use el cifrado en reposo en Amazon DynamoDB.

Todos los datos de la tabla están cifrados

El cifrado en reposo en el lado del servidor está habilitado para los datos de todas las tablas de DynamoDB y no se puede desactivar. No puede cifrar sólo un subconjunto de elementos de una tabla.

El cifrado en reposo solo cifra los datos mientras están estáticos (en reposo) en un medio de almacenamiento persistente. Si le preocupa la seguridad de los datos cuando están en tránsito o en uso, puede ser conveniente adoptar medidas adicionales:

- Datos en tránsito: todos los datos de DynamoDB se cifran en tránsito. De forma predeterminada, las comunicaciones de entrada y salida de DynamoDB usan el protocolo HTTPS, que protege el tráfico de la red mediante el uso del cifrado de Capa de conexión segura (SSL)/Transport Layer Security (TLS).
- Datos en uso: proteja lo datos antes de enviarlos a DynamoDB mediante el cifrado del lado del cliente. Para obtener más información, consulte Cifrado del lado del cliente y del lado del servidor en la Guía para desarrolladores del Cliente de encriptación de Amazon DynamoDB.

Puede usar transmisiones con tablas cifradas. Las transmisiones de DynamoDB siempre se cifran con una clave de cifrado de nivel de tabla. Para obtener más información, consulte Captura de datos de cambios para DynamoDB Streams.

Las copias de seguridad de DynamoDB se cifran y el cifrado también se habilita para la tabla que se restaura a partir de la copia de seguridad. Puede usar la Clave propiedad de AWS, la Clave administrada de AWS o la clave administrada por el cliente para cifrar los datos de la copia de seguridad. Para obtener más información, consulte Copia de seguridad y restauración para DynamoDB.

Los índices secundarios locales y globales se cifran usando la misma clave que la tabla base.

Tipos de cifrado

> **Note**
>
> Las claves administradas por el cliente no se admiten en la versión de 2017 de la tabla global. Si desea utilizar una clave administrada por el cliente en una tabla global de DynamoDB, debe actualizar la tabla a la versión de 2019 de la tabla global y, después, habilitarla.

En la Consola de administración de AWS, el tipo de cifrado es `KMS` cuando se usa la Clave administrada de AWS o la clave administrada por el cliente para cifrar los datos. El tipo de cifrado es `DEFAULT` cuando se usa la Clave propiedad de AWS. En la API de Amazon DynamoDB, el tipo de cifrado es `KMS` cuando se usa la Clave administrada de AWS o la clave administrada por el

cliente. En caso de que no haya un tipo de cifrado, sus datos se cifran utilizando el formato de Clave propiedad de AWS. Puede alternar entre una Clave propiedad de AWS, una Clave administrada de AWS y una clave administrada por el cliente en cualquier momento. Puede utilizar la consola, la AWS Command Line Interface (AWS CLI) o la API de Amazon DynamoDB para alternar entre claves de cifrado.

Tenga en cuenta las siguientes limitaciones al utilizar claves administradas por el cliente:

- No puede utilizar una clave administrada por el cliente con los clústeres de DynamoDB Accelerator (DAX). Para obtener más información, consulte Cifrado en reposo de DAX.
- Puede utilizar una clave administrada por el cliente para cifrar tablas que utilizan transacciones. Sin embargo, para garantizar la durabilidad de la propagación de las transacciones, el servicio almacena temporalmente una copia de la solicitud de transacción y se cifra mediante una Clave propiedad de AWS. Los datos confirmados en sus tablas e los índices secundarios están siempre cifrados en reposo utilizando la clave administrada por el cliente.
- Una clave administrada por el cliente se puede usar para cifrar tablas que utilizan Contributor Insights. Sin embargo, los datos que se transmiten a Amazon CloudWatch están cifrados con una Clave propiedad de AWS.
- Cuando realice la transición a una nueva clave administrada por el cliente, asegúrese de mantener la clave original activada hasta que se complete el proceso. AWS seguirá necesitando la clave original para descifrar los datos antes de cifrarlos con la nueva clave. El proceso se completará cuando el estado de SSEDescription de la tabla sea ENABLED (HABILITADO) y se muestre el KMSMasterKeyArn de la nueva clave administrada por el cliente. A partir de este momento, la clave original se puede desactivar o programar para su eliminación.
- Una vez que se muestra la nueva clave administrada por el cliente, la tabla y cualquier nueva copia de seguridad bajo demanda se cifran con la nueva clave.
- Cualquier copia de seguridad bajo demanda existente permanece cifrada con la clave administrada por el cliente que se utilizó cuando se crearon esas copias de seguridad. Necesitarás esa misma clave para restaurar esas copias de seguridad. Puede identificar la clave del período en que se creó cada copia de seguridad mediante la API DescribeBackup para ver la SSEDescription de esa copia de seguridad.
- Si desactiva su clave administrada por el cliente o la programa para que se elimine, todos los datos de DynamoDB Streams quedarán sujetos a una durabilidad de 24 horas. Todos los datos de actividad que no se hayan recuperado se podrán recortar cuando alcancen una antigüedad superior a 24 horas.

- Si desactiva la clave administrada por el cliente o la programa para su eliminación, las eliminaciones de período de vida (TTL) continuarán durante 30 minutos. Estas eliminaciones de TTL seguirán emitiéndose a DynamoDB Streams y estarán sujetas al intervalo estándar de recorte/retención.

  Para obtener más información, consulte Habilitar claves y Eliminar claves.

Uso de claves KMS y claves de datos

La característica de cifrado en reposo de DynamoDB utiliza una AWS KMS key y una jerarquía de claves de datos para proteger los datos de su tabla. DynamoDB utiliza la misma jerarquía de claves para proteger las secuencias de DynamoDB, las tablas globales y las copias de seguridad cuando se escriben en soportes duraderos.

Le recomendamos que planifique la estrategia de cifrado antes de implementar la tabla en DynamoDB. Si almacena datos confidenciales en DynamoDB, considere incluir el cifrado del cliente en el plan. De esta forma, puede cifrar los datos lo más cerca posible del origen y garantizar la protección durante todo el ciclo de vida. Para obtener más información, consulte la documentación de cliente de cifrado de DynamoDB.

AWS KMS key

El cifrado en reposo protege las tablas de DynamoDB con una AWS KMS key. De forma predeterminada, DynamoDB utiliza una Clave propiedad de AWS, una clave de cifrado de varios inquilinos que se crea y administra en una cuenta de servicio de DynamoDB. Pero puede cifrar las tablas de DynamoDB bajo una clave administrada por el cliente para DynamoDB (`aws/dynamodb`) en la Cuenta de AWS. Puede seleccionar una clave KMS diferente para cada tabla. La clave KMS que seleccione para una tabla también se utiliza para cifrar sus índices secundarios locales y globales, las secuencias y las copias de seguridad.

Al crear o actualizar la tabla, seleccione la clave KMS para una tabla. Puede cambiar la clave KMS de una tabla en cualquier momento, ya sea en la consola de DynamoDB o utilizando la operación UpdateTable. El proceso de cambio de teclas es perfecto y no precisa tiempo de inactividad ni degradación del servicio.

> **Important**
>
> DynamoDB solo admite claves KMS simétricas. No puede utilizar una clave KMS asimétrica para cifrar las tablas de DynamoDB.

Utilice una clave administrada por el cliente para obtener las siguientes características:

- Puede crear y administrar la clave KMS, incluida la configuración de políticas de claves,políticas de IAM y concesiones para controlar el acceso a la clave KMS. Puede habilitar y deshabilitar la clave KMS, habilitar y deshabilitar la rotación de claves automática y eliminar la clave KMS cuando ya no esté en uso.
- Puede utilizar una clave administrada por el cliente con material de claves importado o una clave administrada por el cliente en un almacén de claves personalizado que tenga y administre.
- Puede auditar el cifrado y descifrado de la tabla de DynamoDB examinando las llamadas de la API de DynamoDB a AWS KMS en los registros de AWS CloudTrail.

Utilice la Clave administrada de AWS si necesita alguna de las siguientes características:

- Puede ver la clave KMS y ver su política de claves. (La política de claves no se puede modificar).
- Puede auditar el cifrado y descifrado de la tabla de DynamoDB examinando las llamadas de la API de DynamoDB a AWS KMS en los registros de AWS CloudTrail.

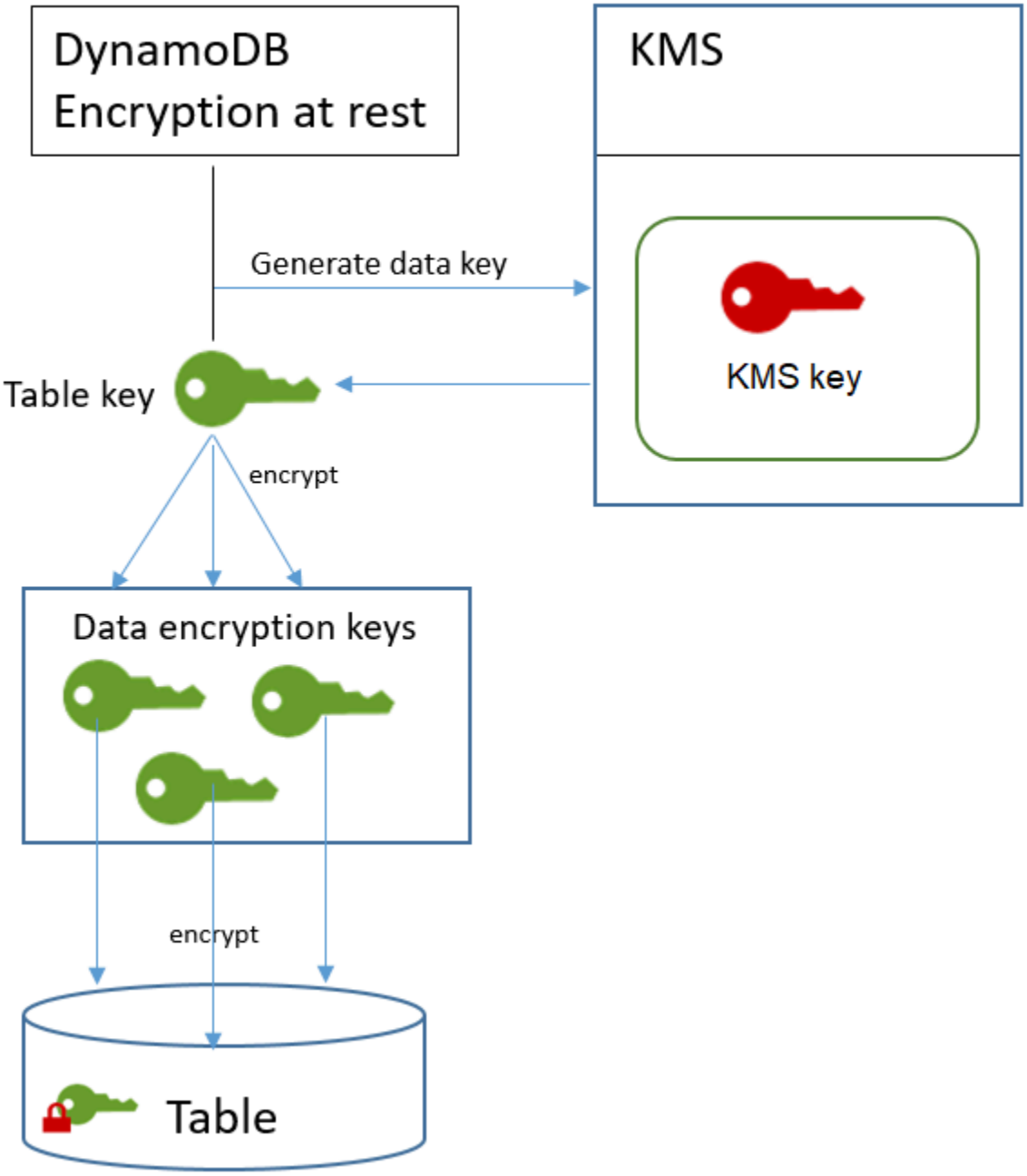
Sin embargo, la Clave propiedad de AWS es gratuita y su uso no se contabiliza en las cuotas de solicitudes o de recursos de AWS KMS. Las claves administradas por el cliente y las Claves administradas por AWS generan cargos por cada llamada a la API y se aplican cuotas de AWS KMS a estas claves KMS.

Claves de tabla

DynamoDB utiliza la clave KMS de la tabla para generar y cifrar una clave de datos única para la tabla, denominada clave de tabla. La clave de tabla se mantiene durante toda la vida útil de la tabla de cifrado.

La clave de la tabla se utiliza como clave de cifrado de claves. DynamoDB utiliza esta clave de tabla para proteger las claves de cifrado de datos que se utilizan para cifrar los datos de la tabla. DynamoDB genera una clave de cifrado de datos única para cada estructura subyacente en una

tabla, pero varios elementos de la tabla pueden protegerse mediante la misma clave de cifrado de datos.



La primera vez que obtiene acceso a una tabla cifrada, DynamoDB envía una solicitud a AWS KMS para utilizar la clave KMS para descifrar la clave de tabla. A continuación, utiliza la clave de tabla de texto sin cifrar para descifrar las claves de cifrado de datos y utiliza las claves de cifrado de datos de texto sin cifrar para descifrar los datos de la tabla.

DynamoDB almacena y utiliza la clave de la tabla y las claves de cifrado de datos fuera de AWS KMS. Protege todas las claves con cifrado Advanced Encryption Standard (AES) y claves de cifrado de 256 bits. A continuación, almacena las claves cifradas con los datos cifrados para que estén disponibles para descifrar los datos de la tabla bajo demanda.

Si cambia la clave de su tabla, DynamoDB genera una nueva clave de tabla. A continuación, utiliza la nueva clave de tabla para volver a cifrar las claves de cifrado de los datos.

Almacenamiento en caché de las claves de tabla

Para evitar llamar a AWS KMS para cada operación de DynamoDB, DynamoDB almacena en la memoria caché las claves de tabla de texto sin cifrar para cada intermediario. Si DynamoDB recibe una solicitud de la clave de tabla almacenada en caché tras cinco minutos de inactividad, envía una nueva solicitud a AWS KMS para descifrar la clave de tabla. Esta llamada capturará los cambios realizados en las políticas de acceso de la clave KMS en AWS KMS o AWS Identity and Access Management (IAM) desde la última solicitud para descifrar la clave de tabla.

Autorizar el uso de su clave KMS

Si utiliza una clave administrada por el cliente o la Clave administrada de AWS en su cuenta para proteger su tabla de DynamoDB, las políticas en esa clave KMS deben conceder permiso a DynamoDB para utilizarla en su nombre. El contexto de autorización de la Clave administrada de AWS para DynamoDB incluye su política de claves y concede a ese delegado los permisos para utilizarla.

Tiene control total sobre las políticas y concesiones de una clave administrada por el cliente debido a que la Clave administrada de AWS está en su cuenta, puede ver sus políticas y concesiones. Sin embargo, como está administrada por AWS, no puede cambiar las políticas.

DynamoDB no necesita autorización adicional para utilizar la Clave propiedad de AWS predeterminada para proteger las tablas de DynamoDB de su cuenta de Cuenta de AWS.

Temas

- Política de claves para una Clave administrada de AWS
- Política de claves para una clave administrada por el cliente
- Usar concesiones para autorizar a DynamoDB

Política de claves para una Clave administrada de AWS

Cuando DynamoDB utiliza la Clave administrada de AWS para DynamoDB (`aws/dynamodb`) en operaciones criptográficas, lo hace en nombre del usuario que obtiene acceso al recurso de DynamoDB. La política de claves de Clave administrada de AWS concede permiso a todos los usuarios de la cuenta para utilizar la Clave administrada de AWS para las operaciones especificadas. Sin embargo, el permiso solo se concede cuando DynamoDB realiza la solicitud en nombre del usuario. La condición ViaService de la política de claves no permite que ningún usuario pueda utilizar la Clave administrada de AWS, a menos que la solicitud se origine con el servicio DynamoDB.

Esta política de claves, como las políticas de todas las Claves administradas por AWS, la establece AWS. No puede cambiarla, pero puede verla en cualquier momento. Para obtener más detalles, consulte Ver una política de clave.

Las declaraciones de política de la política de claves tienen el siguiente efecto:

- Permite a los usuarios de la cuenta utilizar la Clave administrada de AWS para DynamoDB en operaciones criptográficas solo cuando la solicitud proviene de DynamoDB en su nombre. La política también permite a los usuarios crear concesiones para la clave KMS.
- Permite que las identidades de IAM autorizadas en la cuenta vean las propiedades de la Clave administrada de AWS para DynamoDB y para revocar la concesión que permite a DynamoDB utilizar la clave KMS. DynamoDB usa concesiones para las operaciones de mantenimiento en curso.
- Permite a DynamoDB realizar operaciones de solo lectura para buscar la Clave administrada de AWS para DynamoDB en su cuenta.

JSON

```
{
  "Version":"2012-10-17",
  "Id" : "auto-dynamodb-1",
  "Statement" : [ {
    "Sid" : "Allow access through Amazon DynamoDB for all principals in the
 account that are authorized to use Amazon DynamoDB",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "*"
    },
```

```
    "Action" : [ "kms:Encrypt", "kms:Decrypt", "kms:ReEncrypt*",
 "kms:GenerateDataKey*", "kms:CreateGrant", "kms:DescribeKey" ],
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "kms:CallerAccount" : "111122223333",
        "kms:ViaService" : "dynamodb.us-west-2.amazonaws.com"
      }
    }
  }, {
    "Sid" : "Allow direct access to key metadata to the account",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
    },
    "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*", "kms:RevokeGrant" ],
    "Resource" : "*"
  }, {
    "Sid" : "Allow DynamoDB Service with service principal name
 dynamodb.amazonaws.com to describe the key directly",
    "Effect" : "Allow",
    "Principal" : {
      "Service" : "dynamodb.amazonaws.com"
    },
    "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*" ],
    "Resource" : "*"
  } ]
}
```

Política de claves para una clave administrada por el cliente

Cuando selecciona una clave administrada por el cliente para proteger una tabla de DynamoDB, DynamoDB obtiene permiso para utilizar la clave KMS en nombre de la entidad principal que realiza la selección. Esa entidad principal, un usuario o rol, debe tener los permisos en la clave KMS que precisa DynamoDB. Puede proporcionar estos permisos en una política de claves, una política de IAM o una concesión.

Como mínimo, DynamoDB precisa los siguientes permisos en una clave administrada por el cliente:

- kms:Encrypt
- kms:Decrypt

- kms:ReEncrypt* (para kms:ReEncryptFrom y kms:ReEncryptTo)
- kms:GenerateDataKey* (para kms:GenerateDataKey y kms:GenerateDataKeyWithoutPlaintext)
- kms:DescribeKey
- kms:CreateGrant

Por ejemplo, la política de claves de ejemplo siguiente proporciona solo los permisos necesarios. La política tiene las siguientes consecuencias:

- Permite a DynamoDB utilizar la clave KMS en operaciones criptográficas y crear concesiones, pero solo cuando actúa en nombre de las entidades principales de la cuenta que tienen permiso para usar DynamoDB. Si las entidades principales especificadas en la declaración de política no tienen permiso para utilizar DynamoDB, la llamada falla, incluso cuando proviene del servicio de DynamoDB.
- La clave de condición kms:ViaService concede los permisos solo cuando la solicitud proviene de DynamoDB en nombre de las entidades principales enumeradas en la declaración de política. Estas entidades principales no pueden llamar a estas operaciones directamente. Tenga en cuenta que el valor `kms:ViaService`, `dynamodb.*.amazonaws.com`, tiene un asterisco (*) en la posición Región. DynamoDB requiere el permiso para ser independiente de cualquier Región de AWS en particular para que pueda realizar llamadas entre regiones para admitir tablas globales de DynamoDB.
- Proporciona a los administradores de la clave KMS (usuarios que pueden asumir el rol de `db-team`) acceso de solo lectura a la clave KMS y permiso para revocar las concesiones, incluidas las concesiones que DynamoDB precisa para proteger la tabla.

Antes de utilizar una política de claves de ejemplo, sustituya las entidades principales de ejemplo por las entidades principales reales de su cuenta de Cuenta de AWS.

JSON

```
{
  "Id": "key-policy-dynamodb",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid" : "Allow access through Amazon DynamoDB for all principals in the
 account that are authorized to use Amazon DynamoDB",
```

```
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111122223333:user/db-lead"},
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey",
        "kms:CreateGrant"
      ],
      "Resource": "*",
      "Condition": {
         "StringLike": {
           "kms:ViaService" : "dynamodb.*.amazonaws.com"
         }
      }
    },
    {
      "Sid":  "Allow administrators to view the KMS key and revoke grants",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/db-team"
       },
      "Action": [
        "kms:Describe*",
        "kms:Get*",
        "kms:List*",
        "kms:RevokeGrant"
      ],
      "Resource": "*"
    }
  ]
}
```

Usar concesiones para autorizar a DynamoDB

Además de las políticas de claves, DynamoDB utiliza concesiones para establecer permisos en una clave administrada por el cliente o la Clave administrada de AWS para DynamoDB (`aws/dynamodb`). Para ver las concesiones que tiene una clave KMS en su cuenta, utilice la operación ListGrants. DynamoDB no necesita concesiones ni permisos adicionales para utilizar la Clave propiedad de AWS para proteger su tabla.

DynamoDB utiliza los permisos de concesión cuando realiza el mantenimiento del sistema en segundo plano y en tareas de protección de datos continuas. También utiliza las concesiones para generar las claves de la tabla.

Cada concesión es específica de una tabla. Si la cuenta incluye varias tablas cifradas con la misma clave KMS, habrá una concesión de cada tipo para cada tabla. La concesión está limitada por el contexto de cifrado de DynamoDB, que incluye el nombre de la tabla y el ID de la cuenta de Cuenta de AWS e incluye permiso para retirar la concesión si ya no se necesita.

Para crear las concesiones, DynamoDB debe tener permiso para llamar a `CreateGrant` en nombre del usuario que creó la tabla cifrada. Para Claves administradas por AWS, DynamoDB obtiene el permiso `kms:CreateGrant` de la política de claves, que permite a los usuarios de la cuenta llamar a CreateGrant en la clave KMS solo cuando DynamoDB realiza la solicitud en nombre de un usuario autorizado.

La política de claves también puede permitir a la cuenta revocar la concesión en la clave KMS. No obstante, si revoca la concesión en una tabla cifrada activa, DynamoDB no podrá proteger y mantener la tabla.

Contexto de cifrado de DynamoDB

Un contexto de cifrado es un conjunto de pares de clave-valor que contienen datos no secretos arbitrarios. Cuando se incluye un contexto de cifrado en una solicitud para cifrar datos, AWS KMS vincula criptográficamente el contexto de cifrado a los datos cifrados. Para descifrar los datos, es necesario pasar el mismo contexto de cifrado.

DynamoDB utiliza el mismo contexto de cifrado en todas las operaciones criptográficas de AWS KMS. Si utiliza una clave administrada por el cliente o una Clave administrada de AWS para proteger la tabla de DynamoDB, puede utilizar el contexto de cifrado para identificar el uso de la clave KMS en los registros de auditoría. También aparece en texto no cifrado en los registros, como AWS CloudTrail y Amazon CloudWatch Logs.

El contexto de cifrado también se puede utilizar como condición para la autorización en políticas y concesiones. DynamoDB utiliza el contexto de cifrado para restringir las concesiones que permiten el acceso a la clave administrada por el cliente o a Clave administrada de AWS en su cuenta y región.

En su solicitudes a AWS KMS, DynamoDB utiliza un contexto de cifrado con dos parejas de clave-valor.

```
"encryptionContextSubset": {
    "aws:dynamodb:tableName": "Books"
```

```
    "aws:dynamodb:subscriberId": "111122223333"
}
```

- Tabla: el primer par de clave-valor identifica la tabla que está cifrando DynamoDB. La clave es `aws:dynamodb:tableName`. El valor es el nombre de la tabla.

  ```
  "aws:dynamodb:tableName": "<table-name>"
  ```

  Por ejemplo:

  ```
  "aws:dynamodb:tableName": "Books"
  ```

- Cuenta: el segundo par de clave-valor identifica la cuenta de Cuenta de AWS. La clave es `aws:dynamodb:subscriberId`. El valor es el ID de la cuenta.

  ```
  "aws:dynamodb:subscriberId": "<account-id>"
  ```

  Por ejemplo:

  ```
  "aws:dynamodb:subscriberId": "111122223333"
  ```

Supervisión de la interacción de DynamoDB con AWS KMS

Si utiliza una clave administrada por el cliente o una Clave administrada de AWS para proteger las tablas de DynamoDB, puede utilizar los registros de AWS CloudTrail para realizar un seguimiento de las solicitudes que DynamoDB envía a AWS KMS en su nombre.

Las solicitudes `GenerateDataKey`, `Decrypt` y `CreateGrant` se explican en esta sección. Además, DynamoDB utiliza una operación DescribeKey para determinar si la clave KMS que ha seleccionado existe en la cuenta y región. También utiliza una operación RetireGrant para quitar una concesión cuando se elimina una tabla.

GenerateDataKey

Al habilitar el cifrado en reposo en una tabla, DynamoDB crea una clave de tabla única. Envía una solicitud GenerateDataKey a AWS KMS que especifica la clave KMS de la tabla.

El evento que registra la operación `GenerateDataKey` es similar al siguiente evento de ejemplo. El usuario es la cuenta del servicio DynamoDB. Los parámetros incluyen el Nombre de recurso de

Amazon (ARN) de la clave KMS, un especificador de clave que requiere una clave de 256 bits y el contexto de cifrado que identifica la tabla y la cuenta de Cuenta de AWS.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T00:15:17Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:dynamodb:tableName": "Services",
            "aws:dynamodb:subscriberId": "111122223333"
        },
        "keySpec": "AES_256",
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "229386c1-111c-11e8-9e21-c11ed5a52190",
    "eventID": "e3c436e9-ebca-494e-9457-8123a1f5e979",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333",
    "sharedEventID": "bf915fa6-6ceb-4659-8912-e36b69846aad"
}
```

Decrypt

Cuando accede a una tabla de DynamoDB cifrada, DynamoDB necesita descifrar la clave de la tabla de manera que pueda descifrar las claves que hay debajo en la jerarquía. A continuación, descifra los datos de la tabla. Para descifrar la clave de la tabla. DynamoDB envía una solicitud Decrypt a AWS KMS que especifica la clave KMS para la tabla.

El evento que registra la operación `Decrypt` es similar al siguiente evento de ejemplo. El usuario es la entidad principal en su cuenta de Cuenta de AWS que está accediendo a la tabla. Los parámetros incluyen la clave de la tabla cifrada (como blob de texto cifrado) y el contexto de cifrado que identifica la tabla y la cuenta de Cuenta de AWS. AWS KMS deriva el ID de la clave KMS del texto cifrado.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:user01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-02-14T16:42:15Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDT3HGFQZX4RY6RU",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            }
        },
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T16:42:39Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
```

```
    "requestParameters":
    {
        "encryptionContext":
        {
            "aws:dynamodb:tableName": "Books",
            "aws:dynamodb:subscriberId": "111122223333"
        }
    },
    "responseElements": null,
    "requestID": "11cab293-11a6-11e8-8386-13160d3e5db5",
    "eventID": "b7d16574-e887-4b5b-a064-bf92f8ec9ad3",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

CreateGrant

Cuando utiliza una clave administrada por el cliente o una Clave administrada de AWS para proteger la tabla de DynamoDB, DynamoDB utiliza concesiones para permitir al servicio realizar la protección de datos continua y tareas de mantenimiento y durabilidad. Estas concesiones no son obligatorias en las Clave propiedad de AWS.

Las concesiones que crea DynamoDB son específicas de una tabla. El principal en la solicitud CreateGrant es el usuario que creó la tabla.

El evento que registra la operación `CreateGrant` es similar al siguiente evento de ejemplo. Los parámetros incluyen el nombre de recurso de Amazon (ARN) de la clave KMS para la tabla, la entidad principal beneficiaria, la entidad principal de retirada (el servicio de DynamoDB) y las operaciones que cubre la concesión. También incluye una restricción que requiere que toda operación de cifrado utilice el contexto de cifrado especificado.

```
{
    "eventVersion": "1.05",
```

```
    "userIdentity":
    {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:user01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-02-14T00:12:02Z"
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            }
        },
        "invokedBy": "dynamodb.amazonaws.com"
    },
    "eventTime": "2018-02-14T00:15:15Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "dynamodb.amazonaws.com",
    "userAgent": "dynamodb.amazonaws.com",
    "requestParameters": {
        "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "retiringPrincipal": "dynamodb.us-west-2.amazonaws.com",
        "constraints": {
            "encryptionContextSubset": {
                "aws:dynamodb:tableName": "Books",
                "aws:dynamodb:subscriberId": "111122223333"
            }
        },
        "granteePrincipal": "dynamodb.us-west-2.amazonaws.com",
        "operations": [
            "DescribeKey",
            "GenerateDataKey",
            "Decrypt",
            "Encrypt",
            "ReEncryptFrom",
```

```
            "ReEncryptTo",
            "RetireGrant"
        ]
    },
    "responseElements": {
        "grantId":
 "5c5cd4a3d68e65e77795f5ccc2516dff057308172b0cd107c85b5215c6e48bde"
    },
    "requestID": "2192b82a-111c-11e8-a528-f398979205d8",
    "eventID": "a03d65c3-9fee-4111-9816-8bf96b73df01",
    "readOnly": false,
    "resources": [
        {
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```


Administración de tablas de cifrado en DynamoDB

Puede usar la Consola de administración de AWS o la AWS Command Line Interface (AWS CLI) para especificar la clave de cifrado en las tablas nuevas y actualizar las claves de cifrado de las tablas existentes en Amazon DynamoDB.

Temas

- Especificación de la clave de cifrado para una tabla nueva
- Configuración de una clave de cifrado

Especificación de la clave de cifrado para una tabla nueva

Siga estos pasos para especificar la clave de cifrado en una tabla nueva mediante la consola de Amazon DynamoDB o la AWS CLI.

Creación de una tabla cifrada (consola)

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb.
2. En el panel de navegación del lado izquierdo de la consola, elija Tablas.
3. Seleccione Create Table (Crear tabla). En Nombre de la tabla, escriba **`Music`**. Como clave principal, introduzca **`Artist`** y como clave de ordenación, introduzca **`SongTitle`**, ambas como cadenas.
4. En Settings (Ajustes), asegúrese de que Customize settings (Personalizar ajustes) esté seleccionado.

   > **Note**
   >
   > Si selecciona Use default settings (Usar configuración predeterminada), las tablas se cifrarán en reposo con la Clave propiedad de AWS sin costo adicional.

5. En Encryption at rest (Cifrado en reposo), elija un tipo de cifrado: Clave propiedad de AWS, Clave administrada de AWS o una clave administrada por el cliente.

   - Owned by Amazon DynamoDB (Propiedad de Amazon DynamoDB). Clave propiedad de AWS, propiedad y administrada específicamente por DynamoDB. No se cobrará ningún cargo adicional por el uso de esta clave.
   - AWS Clave administrada por. Alias de clave: `aws/dynamodb`. La clave se almacena en su cuenta y la administra AWS Key Management Service (AWS KMS). Se aplican los cargos de AWS KMS.
   - Además de estar almacenada en su cuenta, es de su propiedad y está administrada por usted. Clave administrada por clientes. La clave se almacena en su cuenta y la administra AWS Key Management Service (AWS KMS). Se aplican los cargos de AWS KMS.

     > **Note**
     >
     > Si decide ser propietario y administrar su propia clave, asegúrese de que la política de claves KMS esté configurada correctamente. Para obtener más información, incluidos ejemplos, consulte Política de claves para una clave administrada por el cliente.

6. Elija Create table (Crear tabla) para crear la tabla cifrada. Para confirmar el tipo de cifrado, seleccione los detalles de la tabla en la pestaña Overview (Resumen) y revise la sección Additional details (Detalles adicionales).

Creación de una tabla cifrada (AWS CLI)

Utilice la AWS CLI para crear una tabla con la Clave propiedad de AWS predeterminada, la Clave administrada de AWS o una clave administrada por el cliente para Amazon DynamoDB.

Para crear una tabla cifrada mediante la predeterminada Clave propiedad de AWS

- Cree la tabla cifrada `Music` como sigue:

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5
```

> **Note**
>
> Esta tabla está ahora cifrada utilizando la Clave propiedad de AWS predeterminada en la cuenta del servicio DynamoDB.

Creación de una tabla cifrada mediante la Clave administrada de AWS para DynamoDB

- Cree la tabla cifrada `Music` como sigue:

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
```

```
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5 \
  --sse-specification Enabled=true,SSEType=KMS
```

El estado `SSEDescription` de la descripción de la tabla se establece en `ENABLED` y `SSEType` es `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Creación de una tabla cifrada mediante la clave administrada por el cliente para DynamoDB

- Cree la tabla cifrada `Music` como sigue:

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
      AttributeName=Artist,AttributeType=S \
      AttributeName=SongTitle,AttributeType=S \
  --key-schema \
      AttributeName=Artist,KeyType=HASH \
      AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
      ReadCapacityUnits=10,WriteCapacityUnits=5 \
  --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-
a123-ab1234a1b234
```

El estado `SSEDescription` de la descripción de la tabla se establece en `ENABLED` y `SSEType` es `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
```

```
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Configuración de una clave de cifrado

También puede usar la consola de DynamoDB o la AWS CLI para actualizar las claves de cifrado de una tabla existente entre una Clave propiedad de AWS, una Clave administrada de AWS y una clave administrada por el cliente en cualquier momento.

Actualización de una clave de cifrado (consola)

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb.
2. En el panel de navegación del lado izquierdo de la consola, elija Tablas.
3. Elija la tabla que desea actualizar.
4. Seleccione el menú desplegable Actions (Acciones) y, a continuación, seleccione la opción Update settings (Actualizar ajustes).
5. Vaya a la pestaña Additional settings (Ajustes adicionales).
6. En Encryption (Cifrado), elija Manage encryption (Administración del cifrado).
7. Elija un tipo de cifrado:
   - Propiedad de Amazon DynamoDB. La clave AWS KMS es propiedad de DynamoDB y este servicio se encarga de su administración. No se cobrará ningún cargo adicional por el uso de esta clave.
   - Clave administrada de AWS Alias de clave: `aws/dynamodb`. La clave se almacena en su cuenta y la administra AWS Key Management Service. (AWS KMS). Se aplican los cargos de AWS KMS.
   - Además de estar almacenada en su cuenta, es de su propiedad y está administrada por usted. La clave se almacena en su cuenta y la administra AWS Key Management Service. (AWS KMS). Se aplican los cargos de AWS KMS.

> **Note**
>
> Si decide ser propietario y administrar su propia clave, asegúrese de que la política de claves KMS esté configurada correctamente. Para obtener más información, consulte Política de claves para una clave administrada por el cliente.

A continuación, elija Save (Guardar) para actualizar la tabla cifrada. Para confirmar el tipo de cifrado, verifique los detalles de la tabla en la pestaña Overview (Información general).

Actualización de una clave de cifrado (AWS CLI)

Los siguientes ejemplos muestran cómo actualizar una tabla cifrada con la AWS CLI.

Actualización de una tabla cifrada mediante la predeterminada Clave propiedad de AWS

- Actualice la tabla cifrada `Music`, como en el siguiente ejemplo.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=false
```

> **Note**
>
> Esta tabla está ahora cifrada utilizando la Clave propiedad de AWS predeterminada en la cuenta del servicio DynamoDB.

Actualización de una tabla cifrada mediante la Clave administrada de AWS para DynamoDB

- Actualice la tabla cifrada `Music`, como en el siguiente ejemplo.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=true
```

El estado `SSEDescription` de la descripción de la tabla se establece en `ENABLED` y `SSEType` es `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Actualización de una tabla cifrada con una clave administrada por el cliente para DynamoDB

- Actualice la tabla cifrada `Music`, como en el siguiente ejemplo.

```
aws dynamodb update-table \
  --table-name Music \
  --sse-specification Enabled=true,SSEType=KMS,KMSMasterKeyId=abcd1234-abcd-1234-
a123-ab1234a1b234
```

El estado `SSEDescription` de la descripción de la tabla se establece en `ENABLED` y `SSEType` es `KMS`.

```
"SSEDescription": {
  "SSEType": "KMS",
  "Status": "ENABLED",
  "KMSMasterKeyArn": "arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-
a123-ab1234a1b234",
}
```

Protección de las conexiones de DynamoDB mediante puntos de conexión de VPC y políticas de IAM"

Las conexiones están protegidas entre Amazon DynamoDB y las aplicaciones locales y entre DynamoDB y otros recursos de AWS dentro de la misma región de AWS.

Política necesaria para puntos de conexión

Amazon DynamoDB proporciona una API DescribeEndpoints que le permite mostrar la información de los puntos de conexión regionales. Para las solicitudes a los puntos de conexión de DynamoDB públicos, la API responde sin tener en cuenta la política de IAM de DynamoDB configurada,

aunque haya una denegación explícita o implícita en la política de IAM o de punto de conexión de VPC. Esto se debe a que DynamoDB omite de forma intencionada la autorización de la API `DescribeEndpoints`.

Para las solicitudes de un punto de conexión de VPC, tanto las políticas de punto de conexión de IAM como las de la nube privada virtual (VPC) deben autorizar la llamada a la API `DescribeEndpoints` para las entidades principales de administración de identidades y accesos (IAM) solicitantes mediante la acción `dynamodb:DescribeEndpoints` de IAM. De lo contrario, se le denegará el acceso a la API `DescribeEndpoints`.

A continuación, se muestra un ejemplo de una política de puntos de conexión.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "dynamodb:DescribeEndpoints",
            "Resource": "*"
        }
    ]
}
```

Tráfico entre el servicio y las aplicaciones y clientes locales

Tiene dos opciones de conectividad entre su red privada y AWS:

- Una conexión de AWS Site-to-Site VPN. Para obtener más información, consulte ¿Qué es AWS Site-to-Site VPN? en la Guía del usuario de AWS Site-to-Site VPN.
- Una conexión de Direct Connect. Para obtener más información, consulte ¿Qué es Direct Connect? en la Guía del usuario de Direct Connect.

El acceso a DynamoDB a través de la red se realiza mediante las API publicadas por AWS. Los clientes deben admitir el protocolo de seguridad de la capa de transporte (TLS) 1.2.

Nosotros recomendamos TLS 1.3. Los clientes también deben admitir conjuntos de cifrado con confidencialidad directa total (PFS) tales como Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos. Además, debe firmar las solicitudes con un ID de clave de acceso y una clave de acceso secreta que estén asociados a una entidad principal de IAM, o bien, puede usar AWS Security Token Service (STS) para generar credenciales de seguridad temporales a la hora de firmar solicitudes.

Tráfico entre recursos de AWS en la misma región

Un punto de enlace de la Virtual Private Cloud (Amazon VPC) para DynamoDB es una entidad lógica dentro de una VPC que permite la conectividad solo a DynamoDB. La Amazon VPC enruta las solicitudes a DynamoDB y vuelve a enrutar las respuestas a la VPC. Para obtener más información, consulte Puntos de enlace de la VPC en la Guía del usuario de Amazon VPC. Para consultar ejemplos de políticas que puede usar para controlar el acceso desde los puntos de conexión de VPC, consulte Uso de políticas de IAM para controlar el acceso a DynamoDB.

> **Note**
>
> No se puede acceder a los puntos de conexión de VPC de Amazon a través de AWS Site-to-Site VPN o Direct Connect.

AWS Identity and Access Management (IAM) y DynamoDB

AWS Identity and Access Management es un servicio de AWS que ayuda a un administrador a controlar de forma segura el acceso a los recursos de AWS. Los administradores controlan quién está autenticado (ha iniciado sesión) y autorizado (tiene permisos) para utilizar recursos de Amazon DynamoDB y Acelerador de DynamoDB. Puede utilizar IAM para administrar los permisos de acceso e implementar políticas de seguridad para Amazon DynamoDB y Acelerador de DynamoDB. IAM es un servicio de AWS que puede utilizar sin cargo adicional.

Temas

- Identity and Access Management en Amazon DynamoDB
- Uso de condiciones de las políticas de IAM para control de acceso preciso

Identity and Access Management en Amazon DynamoDB

AWS Identity and Access Management (IAM) es un Servicio de AWS que ayuda a los administradores a controlar de forma segura el acceso a los recursos de AWS. Los administradores de IAM controlan quién puede estar autenticado (ha iniciado sesión) y autorizado (tiene permisos) para utilizar los recursos de DynamoDB. IAM es un servicio de Servicio de AWS que se puedes utilizar sin cargo adicional.

Temas

- Público
- Autenticación con identidades
- Administración de acceso mediante políticas
- Cómo funciona Amazon DynamoDB con IAM
- Ejemplos de políticas basadas en identidad de Amazon DynamoDB
- Solución de problemas de identidad y acceso de Amazon DynamoDB
- Política de IAM para evitar la compra de capacidad reservada de DynamoDB

Público

El uso de AWS Identity and Access Management (IAM) varía en función de su rol:

- Usuario del servicio: solicite permisos a su administrador si no puede acceder a las características (consulte Solución de problemas de identidad y acceso de Amazon DynamoDB).
- Administrador del servicio: determine el acceso de los usuarios y envíe las solicitudes de permiso (consulte Cómo funciona Amazon DynamoDB con IAM)
- Administrador de IAM: escriba políticas para administrar el acceso (consulte Ejemplos de políticas basadas en identidad de Amazon DynamoDB).

Autenticación con identidades

La autenticación es la manera de iniciar sesión en AWS mediante credenciales de identidad. Debe estar autenticado como el Usuario raíz de la cuenta de AWS, como un usuario de IAM o asumiendo un rol de IAM.

Puede iniciar sesión como identidad federada con credenciales de un origen de identidad, como AWS IAM Identity Center (IAM Identity Center), autenticación de inicio de sesión único o credenciales de Google/Facebook. Para obtener más información sobre el inicio de sesión, consulte Cómo iniciar sesión en su Cuenta de AWS en la Guía del usuario de AWS Sign-In.

Para el acceso programático, AWS proporciona un SDK y una CLI para firmar criptográficamente las solicitudes. Para obtener más información, consulte AWS Signature Version 4 para solicitudes de API en la Guía del usuario de IAM.

Usuario raíz de la Cuenta de AWS

Cuando se crea una Cuenta de AWS, se comienza con una identidad de inicio de sesión denominada usuario raíz de la Cuenta de AWS que tiene acceso completo a todos Servicios de AWS y recursos. Recomendamos encarecidamente que no utilice el usuario raíz para sus tareas diarias. Para ver las tareas que requieren credenciales de usuario raíz, consulte Tareas que requieren credenciales de usuario raíz en la Guía del usuario de IAM.

Identidad federada

Como práctica recomendada, exija a los usuarios humanos que utilicen la federación con un proveedor de identidades para acceder a Servicios de AWS con credenciales temporales.

Una identidad federada es un usuario de su directorio empresarial, proveedor de identidad web o Directory Service que accede a los Servicios de AWS mediante credenciales de un origen de identidad. Las identidades federadas asumen roles que proporcionan credenciales temporales.

Para una administración de acceso centralizada, recomendamos AWS IAM Identity Center. Para obtener más información, consulte ¿Qué es el Centro de identidades de IAM? en la Guía del usuario de AWS IAM Identity Center.

Usuarios y grupos de IAM

Un usuario de IAM es una identidad que dispone de permisos específicos para una sola persona o aplicación. Recomendamos usar credenciales temporales en lugar de usuarios de IAM con credenciales a largo plazo. Para obtener más información, consulte Solicitar que los usuarios humanos utilicen la federación con un proveedor de identidades para acceder a AWS mediante credenciales temporales en la Guía del usuario de IAM.

Un grupo de IAM especifica una colección de usuarios de IAM y facilita la administración de permisos para grandes conjuntos de usuarios. Para obtener más información, consulte Casos de usos para usuarios de IAM en la Guía del usuario de IAM.

Roles de IAM

Un rol de IAM es una identidad con permisos específicos que proporciona credenciales temporales. Puede asumir un rol cambiando de una función de usuario a un rol de IAM (consola) o llamando a una operación de AWS CLI o AWS. Para obtener más información, consulte Métodos para asumir un rol en la Guía del usuario de IAM.

Los roles de IAM son útiles para el acceso de usuarios federados, los permisos de usuarios de IAM temporales, el acceso entre cuentas, el acceso entre servicios y las aplicaciones que se ejecutan en Amazon EC2. Para obtener más información, consulte Acceso a recursos entre cuentas en IAM en la Guía del usuario de IAM.

Administración de acceso mediante políticas

Para controlar el acceso en AWS, se crean políticas y se adjuntan a identidades o recursos de AWS. Una política define los permisos cuando se asocia a una identidad o un recurso, define sus permisos. AWS evalúa estas políticas cuando una entidad principal realiza una solicitud. La mayoría de las políticas se almacenan en AWS como documentos JSON. Para obtener más información sobre los documentos de política JSON, consulte Información general de políticas de JSON en la Guía del usuario de IAM.

Mediante las políticas, los administradores especifican quién tiene acceso a qué, definiendo qué entidad principal puede realizar acciones sobre qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Un administrador de IAM crea políticas de IAM y las agrega a roles, que los usuarios pueden asumir posteriormente. Las políticas de IAM definen permisos independientemente del método que se utilice para realizar la operación.

Políticas basadas en identidades

Las políticas basadas en identidad son documentos de política de permisos JSON que asocia a una identidad (usuario, grupo o rol). Estas políticas controlan qué acciones pueden realizar las identidades, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte Creación de políticas de IAM en la Guía del usuario de IAM.

Las políticas basadas en la identidad pueden ser políticas insertadas (incrustadas directamente en una sola identidad) o políticas administradas (políticas independientes asociadas a varias identidades). Para obtener información sobre cómo elegir entre políticas administradas e insertadas, consulte Elegir entre políticas administradas y políticas insertadas en la Guía del usuario de IAM.

Políticas basadas en recursos

Las políticas basadas en recursos son documentos de políticas JSON que se asocian a un recurso. Los ejemplos incluyen políticas de confianza de roles de IAM y políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico. Debe especificar una entidad principal en una política en función de recursos.

Las políticas basadas en recursos son políticas insertadas que se encuentran en ese servicio. No se puede utilizar políticas de IAM administradas por AWS en una política basada en recursos.

Otros tipos de políticas

AWS admite tipos de políticas adicionales que pueden establecer el máximo de permisos concedidos por los tipos de políticas más frecuentes:

- Límites de permisos: establecen los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM. Para obtener más información, consulte Límites de permisos para las entidades de IAM en la Guía del usuario de IAM.
- Políticas de control de servicios (SCP): especifican los permisos máximos para una organización o unidad organizativa en AWS Organizations. Para obtener más información, consulte Políticas de control de servicios en la Guía del usuario de AWS Organizations.
- Políticas de control de recursos (RCP): definen los permisos máximos disponibles para los recursos de las cuentas. Para obtener más información, consulte Políticas de control de recursos (RCP) en la Guía del usuario de AWS Organizations.
- Políticas de sesión: políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal para un rol o un usuario federado. Para obtener más información, consulte Políticas de sesión en la Guía del usuario de IAM.

Varios tipos de políticas

Cuando se aplican varios tipos de políticas a una solicitud, los permisos resultantes son más complicados de entender. Para obtener información acerca de cómo AWS decide si permitir o no una solicitud cuando hay varios tipos de políticas implicados, consulte Lógica de evaluación de políticas en la Guía del usuario de IAM.

Cómo funciona Amazon DynamoDB con IAM

Antes de utilizar IAM para administrar el acceso a DynamoDB, conozca qué características de IAM se pueden utilizar con DynamoDB.

Característica de IAM	Compatibilidad con DynamoDB
Políticas basadas en identidades	Sí
Políticas basadas en recursos	Sí
Acciones de políticas	Sí
Recursos de políticas	Sí
Claves de condición de política	Sí
ACL	No
ABAC (etiquetas en políticas)	Sí
Credenciales temporales	Sí
Permisos de entidades principales	Sí
Roles de servicio	Sí
Roles vinculados al servicio	Sí

Para obtener información general sobre cómo funcionan DynamoDB y otros servicios de AWS con la mayoría de las características de IAM, consulte Servicios de AWS que funcionan con IAM en la Guía del usuario de IAM.

Políticas basadas en identidad de DynamoDB

Compatibilidad con las políticas basadas en identidad: sí

Las políticas basadas en identidad son documentos de políticas de permisos JSON que puede asociar a una identidad, como un usuario de IAM, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué

condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte Creación de políticas de IAM en la Guía del usuario de IAM.

Con las políticas basadas en identidades de IAM, puede especificar las acciones y los recursos permitidos o denegados, así como las condiciones en las que se permiten o deniegan las acciones. Para obtener más información sobre los elementos que puede utilizar en una política de JSON, consulte Referencia de los elementos de las políticas de JSON de IAM en la Guía del usuario de IAM.

Ejemplos de políticas basadas en identidad de DynamoDB

Para ver ejemplos de políticas basadas en identidad de DynamoDB, consulte Ejemplos de políticas basadas en identidad de Amazon DynamoDB.

Políticas basadas en recursos de DynamoDB

Compatibilidad con las políticas basadas en recursos: sí

Las políticas basadas en recursos son documentos de política JSON que se asocian a un recurso. Los ejemplos de políticas basadas en recursos son las políticas de confianza de roles de IAM y las políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico. Para el recurso al que se asocia la política, la política define qué acciones puede realizar una entidad principal especificada en ese recurso y en qué condiciones. Debe especificar una entidad principal en una política en función de recursos. Las entidades principales pueden incluir cuentas, usuarios, roles, usuarios federados o Servicios de AWS.

Para habilitar el acceso entre cuentas, puedes especificar toda una cuenta o entidades de IAM de otra cuenta como la entidad principal de una política en función de recursos. Para obtener más información, consulte Acceso a recursos entre cuentas en IAM en la Guía del usuario de IAM.

Acciones de política para DynamoDB

Compatibilidad con las acciones de políticas: sí

Los administradores pueden utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Action` de una política JSON describe las acciones que puede utilizar para conceder o denegar el acceso en una política. Incluya acciones en una política para conceder permisos y así llevar a cabo la operación asociada.

Para ver una lista de las acciones de DynamoDB, consulte Acciones definidas por Amazon DynamoDB en la Referencia de autorizaciones de servicio.

Las acciones de políticas de DynamoDB utilizan el siguiente prefijo antes de la acción:

```
aws
```

Para especificar varias acciones en una única instrucción, sepárelas con comas.

```
"Action": [
      "aws:action1",
      "aws:action2"
         ]
```

Para ver ejemplos de políticas basadas en identidad de DynamoDB, consulte Ejemplos de políticas basadas en identidad de Amazon DynamoDB.

Recursos de política de DynamoDB

Compatibilidad con los recursos de políticas: sí

Los administradores pueden utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Resource` de la política JSON especifica el objeto u objetos a los que se aplica la acción. Como práctica recomendada, especifique un recurso utilizando el Nombre de recurso de Amazon (ARN). Para las acciones que no admiten permisos por recurso, utilice un carácter comodín (*) para indicar que la instrucción se aplica a todos los recursos.

```
"Resource": "*"
```

Para ver una lista de tipos de recursos y sus ARN de DynamoDB, consulte Tipos de recurso definidos por Amazon DynamoDB en la Referencia de autorizaciones de servicio. Para obtener información acerca de las acciones con las que puede especificar el ARN de cada recurso, consulte Acciones definidas por Amazon DynamoDB.

Para ver ejemplos de políticas basadas en identidad de DynamoDB, consulte Ejemplos de políticas basadas en identidad de Amazon DynamoDB.

Claves de condición de política para DynamoDB

Compatibilidad con claves de condición de políticas específicas del servicio: sí

Los administradores pueden utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Condition` especifica cuándo se ejecutan las instrucciones en función de criterios definidos. Puede crear expresiones condicionales que utilizan operadores de condición, tales como igual o menor que, para que la condición de la política coincida con los valores de la solicitud. Para ver todas las claves de condición globales de AWS, consulte Claves de contexto de condición globales de AWS en la Guía del usuario de IAM.

Para ver una lista de las claves de condición de DynamoDB, consulte Claves de condición para Amazon DynamoDB en la Referencia de autorizaciones de servicio. Para obtener más información acerca de las acciones y los recursos con los que puede utilizar una clave de condición, consulte Acciones definidas por Amazon DynamoDB.

Para ver ejemplos de políticas basadas en identidad de DynamoDB, consulte Ejemplos de políticas basadas en identidad de Amazon DynamoDB.

Listas de control de acceso (ACL) de DynamoDB

Compatibilidad con ACL: no

Las listas de control de acceso (ACL) controlan qué entidades principales (miembros de cuentas, usuarios o roles) tienen permisos para acceder a un recurso. Las ACL son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

Control de acceso basado en atributos (ABAC) con DynamoDB

Admite ABAC (etiquetas en las políticas): sí

El control de acceso basado en atributos (ABAC) es una estrategia de autorización que define permisos en función de atributos denominados etiquetas. Puede asociar etiquetas a entidades de IAM y recursos de AWS y, a continuación, diseñar políticas de ABAC para permitir operaciones cuando la etiqueta de la entidad principal coincida con la etiqueta del recurso.

Para controlar el acceso en función de etiquetas, debe proporcionar información de las etiquetas en el elemento de condición de una política utilizando las claves de condición `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`* o `aws:TagKeys`.

Si un servicio admite las tres claves de condición para cada tipo de recurso, el valor es Sí para el servicio. Si un servicio admite las tres claves de condición solo para algunos tipos de recursos, el valor es Parcial.

Para obtener más información sobre ABAC, consulte Definición de permisos con la autorización de ABAC en la Guía del usuario de IAM. Para ver un tutorial con los pasos para configurar ABAC, consulte Uso del control de acceso basado en atributos (ABAC) en la Guía del usuario de IAM.

Uso de credenciales temporales con DynamoDB

Compatibilidad con credenciales temporales: sí

Las credenciales temporales proporcionan acceso a corto plazo a los recursos de AWS y se crean automáticamente cuando se utiliza la federación o se cambia de rol. AWS recomienda generar credenciales temporales de forma dinámica en lugar de usar claves de acceso a largo plazo. Para obtener más información, consulte Credenciales de seguridad temporales en IAM y Servicios de AWS que funcionan con IAM en la Guía del usuario de IAM.

Permisos de entidades principales entre servicios de DynamoDB

Admite sesiones de acceso directo (FAS): sí

Las sesiones de acceso directo (FAS) utilizan los permisos de la entidad principal para llamar a un Servicio de AWS, combinados con el Servicio de AWS solicitante, para realizar solicitudes a servicios posteriores. Para obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulte Reenviar sesiones de acceso.

Roles de servicio para DynamoDB

Compatibilidad con roles de servicio: sí

Un rol de servicio es un rol de IAM que asume un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. Para obtener más información, consulte Creación de un rol para delegar permisos a un Servicio de AWS en la Guía del usuario de IAM.

> **Warning**
>
> Cambiar los permisos de un rol de servicio podría interrumpir la funcionalidad de DynamoDB. Edite los roles de servicio solo cuando DynamoDB proporcione orientación para hacerlo.

Roles vinculados al servicio para DynamoDB

Admite roles vinculados a servicios: sí

Un rol vinculado a servicios es un tipo de rol de servicio que está vinculado a un Servicio de AWS. El servicio puedes asumir el rol para realizar una acción en su nombre. Los roles vinculados a servicios aparecen en la Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puedes ver, pero no editar, los permisos de los roles vinculados a servicios.

Para más información sobre cómo crear o administrar roles vinculados a servicios, consulta Servicios de AWS que funcionan con IAM. Busque un servicio en la tabla que incluya `Yes` en la columna Rol vinculado a un servicio. Seleccione el vínculo Sí para ver la documentación acerca del rol vinculado a servicios para ese servicio.

Roles vinculados al servicio compatibles en DynamoDB

Roles vinculados al servicio siguiente compatibles en DynamoDB.

- DynamoDB utiliza el rol vinculado al servicio AWSServiceRoleForDynamoDBReplication para la replicación de tablas globales en todas las Regiones de AWS. Consulte Seguridad de las tablas globales de DynamoDB para obtener más información acerca del rol vinculado al servicio AWSServiceRoleForDynamoDBReplication.
- DynamoDB Accelerator (DAX) utiliza el rol vinculado al servicio AWSServiceRoleForDAX para configurar y mantener un clúster de DAX. Consulte the section called “Uso de roles vinculados a servicios para DAX” para obtener más información acerca del rol vinculado al servicio AWSServiceRoleForDAX.

Además de estos roles vinculados al servicio de DynamoDB, DynamoDB utiliza el servicio de escalado automático de aplicaciones para administrar automáticamente la configuración de rendimiento en las tablas de modos de capacidad aprovisionadas. El servicio de escalado automático de aplicaciones utiliza el rol vinculado al servicio AWSServiceRoleForApplicationAutoScaling_DynamoDBTable para administrar la configuración de rendimiento en las tablas de DynamoDB que tienen habilitado el escalado automático. Consulte Roles vinculados a servicios para el escalado automático de aplicaciones para obtener más información.

Ejemplos de políticas basadas en identidad de Amazon DynamoDB

De forma predeterminada, los usuarios y roles no tienen permiso para crear ni modificar recursos de DynamoDB. Un administrador de IAM puede crear políticas de IAM para conceder permisos a los usuarios para realizar acciones en los recursos que necesitan.

Para obtener información acerca de cómo crear una política basada en identidades de IAM mediante el uso de estos documentos de políticas JSON de ejemplo, consulte Creación de políticas de IAM (consola) en la Guía del usuario de IAM.

Para obtener más información sobre las acciones y los tipos de recursos definidos por DynamoDB, incluido el formato de los ARN para cada tipo de recurso, consulte Acciones, recursos y claves de condición de Amazon DynamoDB en la Referencia de autorizaciones de servicio.

Temas

- Prácticas recomendadas sobre las políticas
- Uso de la consola de DynamoDB
- Cómo permitir a los usuarios consultar sus propios permisos
- Uso de las políticas basadas en identidades con Amazon DynamoDB

Prácticas recomendadas sobre las políticas

Las políticas basadas en identidad determinan si alguien puede crear, acceder o eliminar los recursos de DynamoDB de la cuenta. Estas acciones pueden generar costos adicionales para su Cuenta de AWS. Siga estas directrices y recomendaciones al crear o editar políticas basadas en identidades:

- Comienza con las políticas administradas por AWSy continúa con los permisos de privilegio mínimo: a fin de comenzar a conceder permisos a los usuarios y las cargas de tarea, utiliza las políticas administradas por AWS, que conceden permisos para muchos casos de uso comunes. Están disponibles en su Cuenta de AWS. Se recomienda definir políticas gestionadas por el cliente de AWS específicas para sus casos de uso a fin de reducir aún más los permisos. Con el fin de obtener más información, consulte las políticas administradas por AWS o las políticas administradas por AWS para funciones de tarea en la Guía de usuario de IAM.
- Aplique permisos de privilegio mínimo: cuando establezca permisos con políticas de IAM, conceda solo los permisos necesarios para realizar una tarea. Para ello, debe definir las acciones que se pueden llevar a cabo en determinados recursos en condiciones específicas, también conocidos

como permisos de privilegios mínimos. Con el fin de obtener más información sobre el uso de IAM para aplicar permisos, consulte Políticas y permisos en IAM en la Guía del usuario de IAM.

- Utilice condiciones en las políticas de IAM para restringir aún más el acceso: puede agregar una condición a sus políticas para limitar el acceso a las acciones y los recursos. Por ejemplo, puedes escribir una condición de políticas para especificar que todas las solicitudes deben enviarse utilizando SSL. También puedes usar condiciones para conceder acceso a acciones de servicios si se emplean a través de un Servicio de AWS determinado como, por ejemplo, CloudFormation. Para obtener más información, consulte Elementos de la política de JSON de IAM: Condición en la Guía del usuario de IAM.
- Utiliza el analizador de acceso de IAM para validar las políticas de IAM con el fin de garantizar la seguridad y funcionalidad de los permisos: el analizador de acceso de IAM valida políticas nuevas y existentes para que respeten el lenguaje (JSON) de las políticas de IAM y las prácticas recomendadas de IAM. El analizador de acceso de IAM proporciona más de 100 verificaciones de políticas y recomendaciones procesables para ayudar a crear políticas seguras y funcionales. Para más información, consulte Validación de políticas con el Analizador de acceso de IAM en la Guía del usuario de IAM.
- Solicite la autenticación multifactor (MFA): si se encuentra en una situación en la que necesite usuarios raíz o de IAM en su Cuenta de AWS, active la MFA para obtener una mayor seguridad. Para exigir la MFA cuando se invoquen las operaciones de la API, añada condiciones de MFA a sus políticas. Para más información, consulte Acceso seguro a la API con MFA en la Guía del usuario de IAM.

Para obtener más información sobre las prácticas recomendadas de IAM, consulte Prácticas recomendadas de seguridad en IAM en la Guía del usuario de IAM.

Uso de la consola de DynamoDB

Para acceder a la consola de Amazon DynamoDB, debe tener un conjunto mínimo de permisos. Estos permisos deben permitirle enumerar y ver detalles sobre los recursos de DynamoDB en su Cuenta de AWS. Si crea una política basada en identidades que sea más restrictiva que el mínimo de permisos necesarios, la consola no funcionará del modo esperado para las entidades (usuarios o roles) que tengan esa política.

No es necesario conceder permisos mínimos para la consola a los usuarios que solo realizan llamadas a la AWS CLI o a la API de AWS. En su lugar, permite el acceso únicamente a las acciones que coincidan con la operación de API que intentan realizar.

Para asegurarse de que los usuarios y los roles puedan seguir utilizando la consola de DynamoDB, asocie también la política `ConsoleAccess` o `ReadOnly` administrada por AWS de DynamoDB a las entidades. Para obtener más información, consulte Adición de permisos a un usuario en la Guía del usuario de IAM:

Cómo permitir a los usuarios consultar sus propios permisos

En este ejemplo, se muestra cómo podría crear una política que permita a los usuarios de IAM ver las políticas administradas e insertadas que se asocian a la identidad de sus usuarios. Esta política incluye permisos para llevar a cabo esta acción en la consola o mediante programación con la AWS CLI o la API de AWS.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

Uso de las políticas basadas en identidades con Amazon DynamoDB

Este tema cubre el uso de las políticas AWS Identity and Access Management basadas en la identidad (IAM) con Amazon DynamoDB y proporciona ejemplos. Los siguientes ejemplos muestran cómo un administrador de la cuenta puede asociar políticas de permisos a identidades de IAM (es decir, usuarios, grupos y roles) y, por lo tanto, conceder permisos para realizar operaciones en recursos de Amazon DynamoDB.

En las secciones de este tema se explica lo siguiente:

- Permisos de IAM necesarios para usar la consola de Amazon DynamoDB
- Políticas de IAM administradas (predefinidas) de AWS para Amazon DynamoDB
- Ejemplos de políticas administradas por el cliente

A continuación se muestra un ejemplo de una política de permisos.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DescribeQueryScanBooksTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:Query",
                "dynamodb:Scan"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:111122223333:table/Books"
        }
    ]
}
```

La política anterior contiene una instrucción que concede permisos para tres acciones de DynamoDB (`dynamodb:DescribeTable`, `dynamodb:Query` y `dynamodb:Scan`) en una tabla de la región

`us-west-2`AWS, propiedad de la cuenta de AWS especificada por *`account-id`*. El nombre de recurso de Amazon (ARN) del valor `Resource` especifica la tabla a la que se aplican los permisos.

Permisos de IAM necesarios para usar la consola de Amazon DynamoDB

Para trabajar con la consola de DynamoDB, los usuarios deben tener un conjunto mínimo de permisos que les permitan trabajar con los recursos de DynamoDB en su cuenta de AWS. Además de estos permisos de DynamoDB, la consola requiere permisos:

- Permisos de Amazon CloudWatch para mostrar las métricas y los gráficos.
- AWS Data PipelinePermisos de para exportar e importar datos de DynamoDB.
- AWS Identity and Access ManagementPermisos de para obtener acceso a los roles necesarios para importar y exportar.
- Permisos de Amazon Simple Notification Service para recibir notificaciones cada vez que se active una alarma de CloudWatch.
- AWS LambdaPermisos de para procesar registros de DynamoDB Streams.

Si crea una política de IAM que sea más restrictiva que el mínimo de permisos necesarios, la consola no funcionará del modo esperado para los usuarios con esa política de IAM. Para asegurarse de que esos usuarios puedan seguir usando la consola de DynamoDB, asocie también la política administrada `AmazonDynamoDBReadOnlyAccess` AWS al usuario, según se explica en Políticas de IAM administradas (predefinidas) de AWS para Amazon DynamoDB.

No es necesario que conceda permisos mínimos para la consola a los usuarios que solo realizan llamadas a la AWS CLI o a la API de Amazon DynamoDB.

> **Note**
>
> Si hace referencia a un punto de conexión de VPC, también tendrá que autorizar la llamada a la API DescribeEndpoints para las entidades principales de IAM solicitantes mediante la acción de IAM (dynamodb:DescribeEndpoints). Para obtener más información consulte () Política necesaria para puntos de conexión.

Políticas de IAM administradas (predefinidas) de AWS para Amazon DynamoDB

AWS aborda muchos casos de uso comunes proporcionando políticas de IAM independientes creadas y administradas por AWS. Estas políticas administradas por AWS conceden los permisos

necesarios para casos de uso comunes, lo que le evita tener que investigar los permisos que se necesitan. Para más información, consulte Políticas administradas de AWS en la Guía del usuario de IAM.

Las siguientes políticas administradas de AWS, que puede asociar a los usuarios de su cuenta, son específicas de DynamoDB y se agrupan según los escenarios de caso de uso:

- AmazonDynamoDBReadOnlyAccess: concede acceso de solo lectura a los recursos de DynamoDB mediante la Consola de administración de AWS.
- AmazonDynamoDBFullAccess: concede acceso pleno a los recursos de DynamoDB mediante la Consola de administración de AWS.

Para consultar estas políticas de permisos administradas por AWS, inicie sesión en la consola de IAM y busque las políticas específicas.

> **Important**
>
> La práctica recomendada es crear políticas de IAM personalizadas que otorguen privilegio mínimo a los usuarios, los roles o los grupos que las requieran.

Ejemplos de políticas administradas por el cliente

En esta sección, encontrará ejemplos de políticas de usuario que conceden permisos para diversas acciones de DynamoDB. Estas políticas funcionan cuando se utilizan los SDK o la AWS de AWS CLI. Cuando usa la consola, debe conceder permisos adicionales específicos a la consola. Para obtener más información, consulte Permisos de IAM necesarios para usar la consola de Amazon DynamoDB.

> **Note**
>
> En los siguientes ejemplos de políticas se emplea una de las regiones de AWS y que contiene un ID de cuenta y nombres de tablas ficticios.

Ejemplos:

- Política de IAM para conceder permisos a todas las acciones de DynamoDB en una tabla
- Política de IAM para conceder permisos de solo lectura en los elementos de una tabla de DynamoDB

- Política de IAM para conceder acceso a una tabla de DynamoDB específica y sus índices
- Política de IAM para leer, escribir, actualizar y eliminar el acceso en una tabla de DynamoDB
- Política de IAM para separar entornos de DynamoDB en la misma cuenta de AWS
- Política de IAM para evitar la compra de capacidad reservada de DynamoDB
- Política de IAM para conceder acceso de lectura solamente a DynamoDB Streams (no para la tabla)
- Política de IAM para permitir una función AWS Lambda para acceder a registros de DynamoDB Stream
- Política de IAM para el acceso de lectura y escritura a un clúster de DynamoDB Accelerator (DAX)

La Guía del usuario de IAM, incluye tres ejemplos adicionales de DynamoDB:

- Amazon DynamoDB: permite el acceso a una determinada tabla
- Amazon DynamoDB:: permite el acceso a determinadas columnas
- Amazon DynamoDB: permite el acceso de DynamoDB a filas en función de un ID de Amazon Cognito

Política de IAM para conceder permisos a todas las acciones de DynamoDB en una tabla

La siguiente política concede permisos para realizar todas las acciones de DynamoDB en una tabla llamada `Books`. El valor de ARN especificado en `Resource` identifica una tabla de una región de AWS determinada. Si sustituye el nombre de la tabla `Books` en el ARN de `Resource` por un carácter comodín (*), permitirá que se lleven a cabo todas las acciones de DynamoDB en todas las tablas de la cuenta. Considere detenidamente las posibles implicaciones de seguridad antes de utilizar un carácter comodín en esta política o en cualquier otra de IAM.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllAPIActionsOnBooks",
            "Effect": "Allow",
            "Action": "dynamodb:*",
```

```
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

> **Note**
>
> Este es un ejemplo del uso de un carácter comodín (*) para permitir todas las acciones, incluida la administración, las operaciones de datos, el monitoreo y la compra de capacidad reservada de DynamoDB. En su lugar, es una práctica recomendada especificar explícitamente cada acción que se va a conceder y solo lo que necesita ese usuario, rol o grupo.

Política de IAM para conceder permisos de solo lectura en los elementos de una tabla de DynamoDB

La siguiente política de permisos concede permisos para las acciones `GetItem`, `BatchGetItem`, `Scan`, `Query` y `ConditionCheckItem` solo de DynamoDB y, como resultado, establece el acceso de sólo lectura a la tabla `Books`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ReadOnlyAPIActionsOnBooks",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        }
    ]
}
```

Política de IAM para conceder acceso a una tabla de DynamoDB específica y sus índices

La política siguiente concede permisos para acciones de modificación de datos en una tabla de DynamoDB llamada `Books` y todos los índices de esa tabla. Para obtener más información sobre cómo funcionan los índices, consulte Mejora del acceso con índices secundarios en DynamoDB.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AccessTableAllIndexesOnBooks",
            "Effect": "Allow",
            "Action": [
              "dynamodb:PutItem",
              "dynamodb:UpdateItem",
              "dynamodb:DeleteItem",
              "dynamodb:BatchWriteItem",
              "dynamodb:GetItem",
              "dynamodb:BatchGetItem",
              "dynamodb:Scan",
              "dynamodb:Query",
              "dynamodb:ConditionCheckItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*"
            ]
        }
    ]
}
```

Política de IAM para leer, escribir, actualizar y eliminar el acceso en una tabla de DynamoDB

Utilice esta política si necesita permitir que su aplicación cree, lea, actualice y elimine datos en tablas, índices y transmisiones de Amazon DynamoDB. Sustituya el nombre de la región de AWS, el ID de su cuenta y el nombre de la tabla o el carácter comodín (*) en su caso.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DynamoDBIndexAndStreamAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetShardIterator",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/stream/*"
            ]
        },
        {
            "Sid": "DynamoDBTableAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:BatchGetItem",
                "dynamodb:BatchWriteItem",
                "dynamodb:ConditionCheckItem",
                "dynamodb:PutItem",
                "dynamodb:DescribeTable",
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books"
        },
        {
            "Sid": "DynamoDBDescribeLimitsAccess",
            "Effect": "Allow",
            "Action": "dynamodb:DescribeLimits",
            "Resource": [
```

```
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books",
                "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*"
            ]
        }
    ]
}
```

Para expandir esta política para que abarque todas las tablas de DynamoDB en todos las regiones de AWS de esta cuenta, utilice un comodín (*) para el nombre de la tabla y región. Por ejemplo:

```
"Resource":[
               "arn:aws:dynamodb:*:123456789012:table/*",
               "arn:aws:dynamodb:*:123456789012:table/*/index/*"
               ]
```

Política de IAM para separar entornos de DynamoDB en la misma cuenta de AWS

Supongamos que tiene entornos independientes, de tal forma que cada uno de ellos mantiene su propia versión de una tabla denominada `ProductCatalog`. Si crea estas tablas `ProductCatalog` desde la misma cuenta de AWS, trabajar en un entorno podrían afectar al otro entorno, debido al modo en que se configuran los permisos. Por ejemplo, las cuotas en el número de operaciones simultáneas del plano de control (como `CreateTable`) se establecen en el nivel de cuenta de AWS.

Por este motivo, cada acción que se realiza en un entorno reduce el número de operaciones disponibles en el otro entorno. Además, existe el riesgo de que el código en un entorno obtenga por error acceso a las talas del otro entorno.

> **Note**
>
> Si desea separar las cargas de trabajo de producción y de prueba para ayudar a controlar el potencial “radio de explosión” de un evento, la práctica recomendada es crear cuentas de AWS para las cargas de trabajo de pruebas y producción. Para obtener más información, consulteAdministración y separación de cuentas de AWS.

Supongamos, además, que tiene dos desarrolladores, Amit y Alice, que están realizando las pruebas de la tabla `ProductCatalog`. En lugar de que cada desarrollador requiera una cuenta AWS, los desarrolladores pueden compartir la misma cuenta de AWS de prueba. En esta cuenta, puede crear

una copia de la misma tabla para cada desarrollador; por ejemplo, `Alice_ProductCatalog` y `Amit_ProductCatalog`. En este caso, puede crear los usuarios Alice y Amit en la cuenta de AWS que creó para el entorno de pruebas. A continuación, puede conceder permisos a estos usuarios para que lleven a cabo acciones de DynamoDB en las tablas de su propiedad.

Para conceder permisos a estos usuarios IAM, puede realizar uno de los siguientes procedimientos:

- Cree una política independiente para cada usuario y, a continuación, adjunte cada política a su usuario por separado. Por ejemplo, puede adjuntar la siguiente política al usuario Alice para permitirle acceso a todas las acciones de DynamoDB en la tabla `Alice_ProductCatalog`:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllAPIActionsOnAliceTable",
            "Effect": "Allow",
            "Action": [
              "dynamodb:DeleteItem",
              "dynamodb:DescribeContributorInsights",
              "dynamodb:RestoreTableToPointInTime",
              "dynamodb:ListTagsOfResource",
              "dynamodb:CreateTableReplica",
              "dynamodb:UpdateContributorInsights",
              "dynamodb:CreateBackup",
              "dynamodb:DeleteTable",
              "dynamodb:UpdateTableReplicaAutoScaling",
              "dynamodb:UpdateContinuousBackups",
              "dynamodb:TagResource",
              "dynamodb:DescribeTable",
              "dynamodb:GetItem",
              "dynamodb:DescribeContinuousBackups",
              "dynamodb:BatchGetItem",
              "dynamodb:UpdateTimeToLive",
              "dynamodb:BatchWriteItem",
              "dynamodb:ConditionCheckItem",
              "dynamodb:UntagResource",
              "dynamodb:PutItem",
              "dynamodb:Scan",
              "dynamodb:Query",
              "dynamodb:UpdateItem",
```

```
                "dynamodb:DeleteTableReplica",
                "dynamodb:DescribeTimeToLive",
                "dynamodb:RestoreTableFromBackup",
                "dynamodb:UpdateTable",
                "dynamodb:DescribeTableReplicaAutoScaling",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:DescribeLimits",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
Alice_ProductCatalog/*"
        }
    ]
}
```

Después, puede crear una política distinta con un recurso diferente (la tabla `Amit_ProductCatalog`) para el usuario Amit.

- En vez de adjuntar políticas a cada usuario, puede usar variables de políticas de IAM para escribir una sola política y adjuntársela a un grupo. Debe crear un grupo y, en el caso de este ejemplo, agregar a ese grupo a los dos usuarios, Alice y Amit. A continuación, se muestra un ejemplo en el que se conceden permisos para llevar a cabo todas las acciones de DynamoDB en la tabla `${aws:username}_ProductCatalog`. Al evaluar la política, la variable `${aws:username}` se reemplaza por el nombre de usuario del solicitante. Por ejemplo, si Alice envía una solicitud para que se agregue un elemento, la acción solamente se permite si Alice está agregando elementos a la tabla `Alice_ProductCatalog`.

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ActionsOnUserSpecificTable",
            "Effect": "Allow",
            "Action": [
              "dynamodb:PutItem",
              "dynamodb:UpdateItem",
              "dynamodb:DeleteItem",
              "dynamodb:BatchWriteItem",
```

```
                "dynamodb:GetItem",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:ConditionCheckItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
${aws:username}_ProductCatalog"
        },
        {
            "Sid": "AdditionalPrivileges",
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListTables",
                "dynamodb:DescribeTable",
                "dynamodb:DescribeContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/*"
        }
    ]
}
```

> **Note**
>
> Cuando se usan variables de una política de IAM, es preciso especificar explícitamente la versión `2012-10-17` del lenguaje de la política de IAM en la política. La versión predeterminada del lenguaje de la política de IAM (`2008-10-17`) no admite variables de políticas.

En lugar de identificar una tabla específica como un recurso, puede utilizar un carácter comodín (*) para conceder permisos en todas las tablas cuyo nombre lleve antepuesto el nombre del usuario que realiza la solicitud, tal y como se muestra a continuación.

```
"Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/${aws:username}_*"
```

Política de IAM para evitar la compra de capacidad reservada de DynamoDB

Con la capacidad reservada de Amazon DynamoDB, se abona una tarifa inicial única y se adquiere el compromiso de abonar un nivel de uso mínimo, con un ahorro significativo, durante un periodo

concreto. Puede utilizar la Consola de administración de AWS para consultar y adquirir capacidad reservada. Sin embargo, puede que no sea conveniente que todos los usuarios de la organización puedan comprar capacidad reservada. Para obtener más información acerca de la capacidad reservada, consulte Precios de Amazon DynamoDB.

DynamoDB ofrece las siguientes operaciones de API para controlar el acceso a la administración de la capacidad reservada:

- `dynamodb:DescribeReservedCapacity`: devuelve las adquisiciones de capacidad reservada que se encuentran en vigor.
- `dynamodb:DescribeReservedCapacityOfferings` – devuelve detalles sobre los planes de capacidad reservada que ofrece en ese momento AWS.
- `dynamodb:PurchaseReservedCapacityOfferings`: lleva a cabo la adquisición propiamente dicha de la capacidad reservada.

La Consola de administración de AWS utiliza estas operaciones de API para mostrar información sobre la capacidad reservada y realizar adquisiciones. No puede llamar a estas operaciones desde un programa de aplicación, ya que solo son accesibles desde la consola. Sin embargo, sí puede permitir o denegar el acceso a estas operaciones en una política de permisos de IAM.

La siguiente política permite a los usuarios ver las compras de capacidad reservada y las ofertas mediante la Consola de administración de AWS, pero no permite llevar a cabo nuevas compras.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReservedCapacityDescriptions",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeReservedCapacity",
                "dynamodb:DescribeReservedCapacityOfferings"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        },
        {
            "Sid": "DenyReservedCapacityPurchases",
```

```
            "Effect": "Deny",
            "Action": "dynamodb:PurchaseReservedCapacityOfferings",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        }
    ]
}
```

Tenga en cuenta que esta política utiliza el carácter comodín (*) para permitir la descripción de permisos para todos los, y para denegar la compra de capacidad reservada de DynamoDB para todos.

Política de IAM para conceder acceso de lectura solamente a DynamoDB Streams (no para la tabla)

Cuando se habilita DynamoDB Streams en una tabla, la información sobre cada modificación de los elementos de datos de esa tabla es capturada. Para obtener más información, consulte Captura de datos de cambios para DynamoDB Streams.

En algunos casos, es posible que desee impedir que una aplicación lea datos desde una tabla de DynamoDB, pero al mismo tiempo desee permitir que se pueda obtener acceso a la secuencia de esa tabla. Por ejemplo, puede configurar AWS Lambda para que sondee la trnasmisión e invoque una función Lambda cuando se detecten actualizaciones de elementos y, a continuación, realice un procesamiento adicional.

Existen las siguientes acciones disponibles para controlar el acceso a DynamoDB streams:

- `dynamodb:DescribeStream`
- `dynamodb:GetRecords`
- `dynamodb:GetShardIterator`
- `dynamodb:ListStreams`

En el ejemplo siguiente se crea una política que concede a los usuarios permisos para acceder a las secuencias de una tabla denominada `GameScores`. El carácter comodín (*) en el ARN permite obtener todos los identificadores de transmisión asociados a esa tabla.

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AccessGameScoresStreamOnly",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/*"
        }
    ]
}
```

Observe que esta política permite acceder a las transmisión de la tabla `GameScores`, pero no a la tabla en sí.

Política de IAM para permitir una función AWS Lambda para acceder a registros de DynamoDB Stream

Si desea llevar a acabo determinadas acciones basándose en los nuevos eventos de una transmisión de DynamoDB, puede escribir una función AWS Lambda que sea activada por esos nuevos eventos. Una función Lambda de este tipo requiere permisos para leer los datos de la transmisión de DynamoDB. Para obtener más información sobre cómo usar Lambda con DynamoDB Streams, consulte DynamoDB Streams y disparadores de AWS Lambda.

Para conceder permisos a Lambda, utilice la política de permisos que está asociada con el rol de IAM de la función Lambda (también conocen como un rol de ejecución). Especifique esta política cuando cree la función Lambda.

Por ejemplo, puede asociar la siguiente política de permisos al rol de ejecución con el fin de conceder permisos a Lambda para realizar las acciones de DynamoDB Streams indicadas.

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "APIAccessForDynamoDBStreams",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/
GameScores/stream/*"
        }
    ]
}
```

Para obtener más información, consulte permisos de AWS Lambda en la Guía para desarrolladores de AWS Lambda.

Política de IAM para el acceso de lectura y escritura a un clúster de DynamoDB Accelerator (DAX)

La siguiente política permite acceder a la lectura, escritura, actualización y eliminación a un clúster de DynamoDB Accelerator (DAX), pero no a la tabla de DynamoDB asociada. Para usar esta política, sustituya el nombre de la región de AWS, el ID de su cuenta y el nombre de su clúster de DAX.

> **Note**
>
> Esta política concede acceso a un }clúster de DAX, pero no a la tabla de DynamoDB asociada. Asegúrese de que su clúster DAX tiene la política correcta para realizar estas mismas operaciones en la tabla de DynamoDB en su nombre.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonDynamoDBDAXDataOperations",
            "Effect": "Allow",
```

```
            "Action": [
                "dax:GetItem",
                "dax:PutItem",
                "dax:ConditionCheckItem",
                "dax:BatchGetItem",
                "dax:BatchWriteItem",
                "dax:DeleteItem",
                "dax:Query",
                "dax:UpdateItem",
                "dax:Scan"
            ],
            "Resource": "arn:aws:dax:eu-west-1:123456789012:cache/MyDAXCluster"
        }
    ]
}
```

Para ampliar esta política a fin de abarcar el acceso a DAX para todas las regiones de AWS de una cuenta, use un carácter comodín (*) para el nombre de la región.

```
"Resource": "arn:aws:dax:*:123456789012:cache/MyDAXCluster"
```

Solución de problemas de identidad y acceso de Amazon DynamoDB

Utilice la siguiente información para diagnosticar y solucionar los problemas comunes que puedan surgir cuando trabaje con DynamoDB e IAM.

Temas

- No tengo autorización para realizar una acción en DynamoDB
- No tengo autorización para realizar la operación iam:PassRole
- Quiero permitir a personas externas a mi Cuenta de AWS el acceso a mis recursos de DynamoDB

No tengo autorización para realizar una acción en DynamoDB

Si la Consola de administración de AWS le indica que no está autorizado para llevar a cabo una acción, debe ponerse en contacto con su administrador para recibir ayuda. Su administrador es la persona que le facilitó su nombre de usuario y contraseña.

En el siguiente ejemplo, el error se produce cuando el usuario `mateojackson` intenta utilizar la consola para consultar los detalles acerca de un recurso ficticio *`my-example-widget`*, pero no tiene los permisos ficticios `aws:`*`GetWidget`*.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 aws:GetWidget on resource: my-example-widget
```

En este caso, Mateo pide a su administrador que actualice sus políticas de forma que pueda obtener acceso al recurso *`my-example-widget`* mediante la acción `aws:`*`GetWidget`*.

No tengo autorización para realizar la operación iam:PassRole

Si recibe un error que indica que no tiene autorización para realizar la acción `iam:PassRole`, las políticas deben actualizarse a fin de permitirle pasar un rol a DynamoDB.

Algunos Servicios de AWS le permiten transferir un rol existente a dicho servicio en lugar de crear un nuevo rol de servicio o uno vinculado al servicio. Para ello, debe tener permisos para transferir el rol al servicio.

En el siguiente ejemplo, el error se produce cuando un usuario de IAM denominado `marymajor` intenta utilizar la consola para realizar una acción en DynamoDB. Sin embargo, la acción requiere que el servicio cuente con permisos que otorguen un rol de servicio. Mary no tiene permisos para transferir el rol al servicio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

En este caso, las políticas de Mary se deben actualizar para permitirle realizar la acción `iam:PassRole`.

Si necesita ayuda, póngase en contacto con su gestionador de AWS. El gestionador es la persona que le proporcionó las credenciales de inicio de sesión.

Quiero permitir a personas externas a mi Cuenta de AWS el acceso a mis recursos de DynamoDB

Puede crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. Puede especificar una persona de confianza para que asuma el rol. En el caso de los servicios que admitan las políticas basadas en recursos o las listas de control de acceso (ACL), puede utilizar dichas políticas para conceder a las personas acceso a sus recursos.

Para obtener más información, consulte lo siguiente:

- Para obtener información acerca de si DynamoDB admite estas características, consulte Cómo funciona Amazon DynamoDB con IAM.
- Para obtener información acerca de cómo proporcionar acceso a los recursos de las Cuentas de AWS de su propiedad, consulte Acceso para un usuario de IAM en otra Cuenta de AWS propia en la Guía del usuario de IAM.
- Para obtener información acerca de cómo proporcionar acceso a tus recursos a Cuentas de AWS de terceros, consulta Proporcionar acceso a Cuentas de AWS que son propiedad de terceros en la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso mediante una federación de identidades, consulta Proporcionar acceso a usuarios autenticados externamente (identidad federada) en la Guía del usuario de IAM.
- Para obtener información acerca del uso de roles y políticas basadas en recursos para el acceso entre cuentas, consulte Acceso a recursos entre cuentas en IAM en la Guía del usuario de IAM.

Política de IAM para evitar la compra de capacidad reservada de DynamoDB

Con la capacidad reservada de Amazon DynamoDB, se abona una tarifa inicial única y se adquiere el compromiso de abonar un nivel de uso mínimo, con un ahorro significativo, durante un periodo concreto. Puede utilizar la Consola de administración de AWS para consultar y adquirir capacidad reservada. Sin embargo, puede que no sea conveniente que todos los usuarios de la organización puedan comprar capacidad reservada. Para obtener más información acerca de la capacidad reservada, consulte Precios de Amazon DynamoDB.

DynamoDB ofrece las siguientes operaciones de API para controlar el acceso a la administración de la capacidad reservada:

- `dynamodb:DescribeReservedCapacity`: devuelve las adquisiciones de capacidad reservada que se encuentran en vigor.
- `dynamodb:DescribeReservedCapacityOfferings` – devuelve detalles sobre los planes de capacidad reservada que ofrece en ese momento AWS.
- `dynamodb:PurchaseReservedCapacityOfferings`: lleva a cabo la adquisición propiamente dicha de la capacidad reservada.

La Consola de administración de AWS utiliza estas operaciones de API para mostrar información sobre la capacidad reservada y realizar adquisiciones. No puede llamar a estas operaciones desde un programa de aplicación, ya que solo son accesibles desde la consola. Sin embargo, sí puede permitir o denegar el acceso a estas operaciones en una política de permisos de IAM.

La siguiente política permite a los usuarios ver las compras de capacidad reservada y las ofertas mediante la Consola de administración de AWS, pero no permite llevar a cabo nuevas compras.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReservedCapacityDescriptions",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeReservedCapacity",
                "dynamodb:DescribeReservedCapacityOfferings"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        },
        {
            "Sid": "DenyReservedCapacityPurchases",
            "Effect": "Deny",
            "Action": "dynamodb:PurchaseReservedCapacityOfferings",
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:*"
        }
    ]
}
```

Tenga en cuenta que esta política utiliza el carácter comodín (*) para permitir la descripción de permisos para todos los, y para denegar la compra de capacidad reservada de DynamoDB para todos.

Uso de condiciones de las políticas de IAM para control de acceso preciso

Al conceder permisos en DynamoDB, puede especificar condiciones que determinan cómo se aplica una política de permisos.

Descripción general

En DynamoDB, existe la opción de especificar condiciones al conceder permisos mediante una política de IAM (consulte Identity and Access Management en Amazon DynamoDB). Por ejemplo, puede hacer lo siguiente:

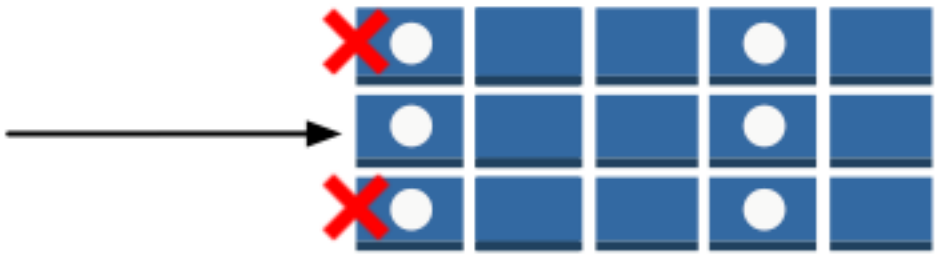
- Conceder permisos para que los usuarios puedan obtener acceso de solo lectura a determinados elementos y atributos de una tabla o un índice secundario.
- Conceder permisos para que los usuarios puedan obtener acceso de solo escritura a determinados atributos de una tabla, según la identidad del usuario en cuestión.

En DynamoDB, puede utilizar las claves de condición para especificar las condiciones en una política de IAM, tal y como se muestra en el caso de uso en la siguiente sección.

Caso de uso de permisos

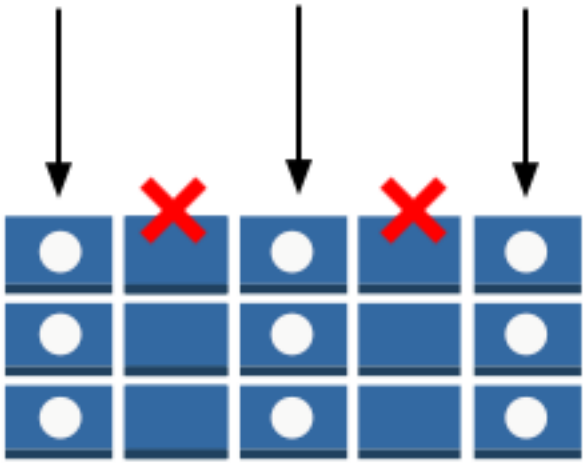
Además de controlar el acceso a acciones de la API de DynamoDB, puede controlar el acceso a elementos y atributos de datos individuales. Por ejemplo, puede hacer lo siguiente:

- Conceder permisos en una tabla, pero restringir el acceso a elementos específicos de esa tabla según los valores de determinadas claves principales. Un ejemplo de ello sería una aplicación de una red social de juegos en la cual los datos de juegos guardados de todos los usuarios se almacenan en una única tabla, pero ningún usuario puede obtener acceso a los elementos de datos que no son de su propiedad, como se muestra en la siguiente ilustración:



- Ocultar la información de tal forma que únicamente un subconjunto de atributos se encuentre visible para el usuario. Un ejemplo de ello sería una aplicación que muestra datos de vuelos en los aeropuertos cercanos a la ubicación donde se encuentra el usuario. Se muestran los nombres de las compañías aéreas, los horarios de llegada y salida y los números de los vuelos. Sin embargo,

se ocultan otros atributos como el nombre de los pilotos o el número de pasajeros, como se muestra en la siguiente ilustración:



Para implementar este tipo de control de acceso preciso, hay que escribir una política de permisos de IAM que especifique las condiciones para obtener acceso a las credenciales de seguridad y los permisos asociados. A continuación, se aplica la política de a los usuarios, los grupos o los roles creados mediante la consola de IAM. La política de IAM puede restringir el acceso a elementos individuales en una tabla, a los atributos de estos elementos o a ambas cosas al mismo tiempo.

Si lo desea, puede usar las identidades web federadas para controlar el acceso de los usuarios cuya autenticación se lleva a cabo mediante Login with Amazon, Facebook o Google. Para obtener más información, consulte Uso de la federación de identidades web.

El componente `Condition` de IAM se utiliza para implementar una política de control de acceso precisa. Puede agregar un componente `Condition` a una política de permisos para permitir o denegar el acceso a los elementos y atributos de las tablas e índices de DynamoDB en función de sus requisitos empresariales concretos.

Por ejemplo, tomemos una aplicación de juegos para móviles que permite a los usuarios seleccionar un juego entre diversas opciones y jugar a él. La aplicación utiliza una tabla de DynamoDB denominada `GameScores` para llevar la cuenta de las puntuaciones altas y otros datos de los usuarios. Cada elemento de la tabla se identifica de forma exclusiva con un identificador de usuario y el nombre del juego que ha utilizado el usuario. La tabla `GameScores` tiene una clave principal compuesta de una clave de partición (`UserId`) y de una clave de ordenación (`GameTitle`). Cada usuario solo tiene acceso a los datos de juego asociados a su propio identificador de usuario. Para

poder jugar, el usuario debe pertenecer a un rol de IAM denominado `GameRole`, que tiene adjunta una política de seguridad.

Para administrar los permisos de los usuarios en esta aplicación, podría escribir una política de permisos como la siguiente:

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAccessToOnlyItemsMatchingUserID",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ],
               "dynamodb:Attributes":[
                  "UserId",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "TopScore",
                  "TopScoreDateTime"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
```

```
        }
      }
   ]
}
```

Además de conceder permisos para acciones de DynamoDB concretas (componente `Action`) en la tabla `GameScores` (componente `Resource`), el componente `Condition` utiliza las claves de condición siguientes específicas de DynamoDB que limitan los permisos como se indica a continuación:

- `dynamodb:LeadingKeys`: esta clave de condición permite a los usuarios obtener acceso solo a los elementos cuyo valor de clave de partición coincide con su ID de usuario. Este identificador, `${www.amazon.com:user_id}`, es una variable de sustitución. Para obtener más información sobre las variables de sustitución, consulte Uso de la federación de identidades web.
- `dynamodb:Attributes`: esta clave de condición limita el acceso a los atributos concretos, de tal forma que solamente las acciones enumeradas en la política de permisos pueden devolver valores para estos atributos. Además, la cláusula `StringEqualsIfExists` garantiza que la aplicación proporcione siempre una lista de atributos específicos que admiten acciones e impide que aplicación pueda solicitar todos los atributos.

Al evaluar una política de IAM, el resultado siempre es true (verdadero, se permite el acceso) o false (falso, se deniega el acceso). Si cualquier parte del componente `Condition` es false, la política completa se evalúa en false y se deniega el acceso.

> **Important**
>
> Si utiliza `dynamodb:Attributes`, debe especificar los nombres de todos los atributos de clave principal y clave de índice de la tabla y también de todos los índices secundarios enumerados en la política. De lo contrario, DynamoDB no podrá utilizar estos atributos de clave para realizar la acción solicitada.

Los documentos de políticas de IAM solamente pueden contener los siguientes caracteres Unicode: tabulador horizontal (U+0009), salto de línea (U+000A), retorno de carro (U+000D) y caracteres comprendidos entre U+0020 y U+00FF.

Especificación de condiciones: uso de claves de condición

AWS proporciona un conjunto de claves de condición predefinidas (las claves de condición generales de AWS), para todos los servicios de AWS que admiten IAM para el control de acceso. Por ejemplo, puede usar la clave de condición `aws:SourceIp` para comprobar la dirección IP del solicitante antes de permitir que se lleve a cabo cualquier acción. Para obtener más información y una lista con las claves generales de AWS, consulte Claves disponibles para condiciones en la Guía del usuario de IAM.

En la siguiente tabla se muestran las claves de condición específicas del servicio DynamoDB que se aplican a DynamoDB.

Clave de condición de DynamoDB	Descripción
`dynamodb:LeadingKeys`	Representa el primer atributo de clave de una tabla; es decir, la clave de partición. El nombre de la clave, `LeadingKeys`, es plural, aunque la clave se utiliza en acciones que afectan a un solo elemento. Además, debe usar el modificador `ForAllValues` cuando utilice `LeadingKeys` en una condición.
`dynamodb:FirstPartitionKeyValues`	Representa el primer atributo de clave de una tabla; es decir, la primera clave de partición. El nombre de la clave, `FirstPartitionKeyValues`, es plural, aunque la clave se utiliza en acciones que afectan a un solo elemento. Además, debe usar el modificador `ForAllValues` cuando utilice `FirstPartitionKeyValues` en una condición. `FirstPartitionKeyValues` y `LeadingKeys` se pueden intercambiar.
`dynamodb:SecondPartitionKeyValues`	Similar a `dynamodb:FirstPartitionKeyValues`. Representa la segunda clave de partición de los recursos. El nombre de la clave, `SecondPartitionKeyValues`, es plural, aunque la clave se utiliza en acciones que afectan a un solo elemento.
`dynamodb:ThirdPartitionKeyValues`	Similar a `dynamodb:FirstPartitionKeyValues`. Represent a la tercera clave de partición de los recursos. El nombre de la clave, `ThirdPartitionKeyValues`, es plural, aunque la clave se utiliza en acciones que afectan a un solo elemento.

Clave de condición de DynamoDB	Descripción
dynamodb:FourthPartitionKeyValues	Similar a dynamodb:FirstPartitionKeyValues. Represent a la cuarta clave de partición de los recursos. El nombre de la clave, FourthPartitionKeyValues, es plural, aunque la clave se utiliza en acciones que afectan a un solo elemento.
dynamodb:Select	Representa el parámetro Select de una solicitud Query o Scan. Select puede ser cualquiera de los valores siguientes: • ALL_ATTRIBUTES • ALL_PROJECTED_ATTRIBUTES • SPECIFIC_ATTRIBUTES • COUNT
dynamodb:Attributes	Representa una lista de los atributos de nivel superior a los que accede una solicitud. Una solicitud accede a un atributo de nivel superior si este o cualquier atributo anidado que contenga, se especifica en los parámetros de la solicitud o se devuelve desde la solicitud. Por ejemplo, si una solicitud GetItem especifica una ProjectionExpression de "Name, Address.City", la lista de dynamodb:Attributes incluiría “Nombre” y “Dirección”.
dynamodb:ReturnValues	Representa el parámetro ReturnValues de una solicitud. ReturnValues puede ser cualquiera de los valores siguientes: • ALL_OLD • UPDATED_OLD • ALL_NEW • UPDATED_NEW • NONE

Clave de condición de DynamoDB	Descripción
`dynamodb:ReturnConsumedCapacity`	Representa el parámetro `ReturnConsumedCapacity` de una solicitud. `ReturnConsumedCapacity` puede ser uno de los valores siguientes: • `TOTAL` • `NONE`

Limitación del acceso de los usuarios

Muchas políticas de permisos de IAM permiten a los usuarios obtener acceso únicamente los elementos de una tabla cuyo valor de clave de partición coincide con el identificador de usuario. Por ejemplo, la aplicación de juegos que hemos mencionado antes limita el acceso de este modo, de tal forma que cada usuario solo pueda obtener acceso a los datos de juego que están asociados a su propio identificador de usuario. Las variables de sustitución `${www.amazon.com:user_id}`, `${graph.facebook.com:id}` y `${accounts.google.com:sub}` de IAM contienen identificadores de usuario de Login with Amazon, Facebook y Google. Para saber cómo una aplicación inicia sesión en uno de estos proveedores de identidad y obtiene estos identificadores, consulte Uso de la federación de identidades web.

Important

No se admite el control de acceso detallado para restringir la replicación de tablas globales. La aplicación de condiciones de política para el control de acceso detallado a entidades principales de servicio o a roles vinculados al servicio de DynamoDB utilizados para la replicación de tablas globales puede interrumpir la replicación en una tabla global.

Note

Cada uno de los ejemplos de la sección siguiente establece en la cláusula `Effect` el valor `Allow` y especifica solamente las acciones, los recursos y los parámetros que se permiten. Únicamente se permite el acceso a aquello que se indica explícitamente en la política de IAM.

En algunos casos, se pueden modificar estas políticas de tal forma que se basen en la denegación (en cuyo caso, se establecería la cláusula `Effect` en `Deny` y se invertiría toda la lógica de la política). Sin embargo, recomendamos evitar el uso de políticas basadas en la denegación con DynamoDB, ya que son difíciles de escribir correctamente en comparación con las políticas basadas en permiso. Además, cualquier cambio futuro de la API de DynamoDB (o de las entradas a la API existente) podrían dejar sin efectividad una política basada en la denegación.

Ejemplos de políticas: uso de condiciones para el control de acceso preciso

En esta sección se muestran varias políticas que permiten implementar el control de acceso preciso en las tablas y los índices de DynamoDB.

Note

Todos los ejemplos utilizan la región us-west-2 y contienen identificadores de cuenta ficticios.

En el siguiente vídeo se explica el control de acceso detallado en DynamoDB utilizando condiciones de políticas de IAM.

1: concesión de permisos que limitan el acceso a los elementos con un valor de clave de partición específico

La siguiente política concede permisos que permiten realizar un conjunto de acciones de DynamoDB en la tabla `GamesScore`. En ella, se utiliza la clave de condición `dynamodb:LeadingKeys` para que los usuarios únicamente puedan realizar acciones en los elementos cuyo valor de clave de partición `UserID` coincida con el identificador de usuario exclusivo de Login with Amazon correspondiente a la aplicación.

Important

La lista de acciones no incluye permisos para `Scan`, porque `Scan` devuelve todos los elementos, sean cuales sean sus claves principales.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"FullAccessToUserItems",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ]
            }
         }
      }
   ]
}
```

> **Note**
>
> Cuando utiliza variables de políticas, debe especificar de forma explícita la versión `2012-10-17` en la política. La versión predeterminada del lenguaje de la política de acceso, `2008-10-17`, no admite variables de políticas.

Para implementar el acceso de solo lectura, puede eliminar todas las acciones que permitan modificar los datos. En la siguiente política, solo se incluyen en la condición las acciones que proporcionan acceso de solo lectura.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ReadOnlyAccessToUserItems",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ]
            }
         }
      }
   ]
}
```

Important

Si utiliza `dynamodb:Attributes`, debe especificar los nombres de todos los atributos de clave principal y clave de índice de la tabla y también de todos los índices secundarios enumerados en la política. De lo contrario, DynamoDB no podrá utilizar estos atributos de clave para realizar la acción solicitada.

2: concesión de permisos que limitan el acceso a determinados atributos de una tabla

La siguiente política de permisos únicamente permite obtener acceso a dos atributos concretos de una tabla; para ello, se agrega la clave de condición `dynamodb:Attributes`. Estos atributos se pueden leer, escribir o evaluar en una escritura condicional o un filtro de examen.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"LimitAccessToSpecificAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem",
            "dynamodb:GetItem",
            "dynamodb:Query",
            "dynamodb:BatchGetItem",
            "dynamodb:Scan"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:Attributes":[
                  "UserId",
                  "TopScore"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES",
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
            }
         }
      }
   ]
```

```
}
```

> **Note**
>
> La política adopta un enfoque de lista blanca que permite obtener acceso a los atributos que se nombran explícitamente. Puede escribir una política equivalente que deniegue el acceso a los demás atributos. No recomendados esta estrategia de lista de denegación. Loa usuarios pueden determinar los nombres de estos atributos denegados por medio del principio de mínimo privilegio que se explica en Wikipedia en la dirección http://en.wikipedia.org/wiki/Principle_of_least_privilege y aplicar un enfoque de lista de permisos para enumerar todos los valores permitidos, en lugar de especificar los atributos denegados.

Esta política no permite `PutItem`, `DeleteItem` ni `BatchWriteItem`. Estas acciones siempre sustituyen el elemento anterior en su totalidad y esto permitiría a los usuarios eliminar valores de atributos anteriores a los que no se les permite obtener acceso.

La cláusula `StringEqualsIfExists` de la política de permisos se asegura de que se cumpla lo siguiente:

- Si el usuario especifica el parámetro `Select`, entonces su valor debe ser `SPECIFIC_ATTRIBUTES`. Este requisito se impide que la acción del API devuelva cualquier atributo que no esté permitido; por ejemplo, desde una proyección de índice.
- Si el usuario especifica el parámetro `ReturnValues`, entonces su valor debe ser `NONE`, `UPDATED_OLD` o `UPDATED_NEW`. Esto es obligatorio porque la acción `UpdateItem` lleva a cabo, además, operaciones de lectura implícitas para comprobar si un elemento existe antes de sustituirlo y para que los valores de atributos anteriores se puedan devolver en caso de que se soliciten. Al restringir `ReturnValues` de este modo, nos aseguramos de que los usuarios solamente puedan leer o escribir los atributos permitidos.
- La cláusula `StringEqualsIfExists` se asegura de que solamente se pueda utilizar uno de estos parámetros (`Select` o `ReturnValues`) en cada solicitud, en el contexto de las acciones permitidas.

A continuación se muestran algunas variaciones de esta política:

- Para permitir exclusivamente acciones de lectura, puede eliminar `UpdateItem` de la lista de acciones permitidas. Dado que ninguna de las acciones restantes aceptan `ReturnValues`, puede eliminar `ReturnValues` de la condición. Además, puede cambiar `StringEqualsIfExists` por `StringEquals`, porque el parámetro `Select` siempre tiene un valor (`ALL_ATTRIBUTES`, a no ser que se especifique otra cosa).
- Para permitir exclusivamente acciones de escritura, puede eliminar todo excepto `UpdateItem` de la lista de acciones permitidas. Dado que `UpdateItem` no utiliza el parámetro `Select`, puede eliminar `Select` de la condición. Además, debe cambiar `StringEqualsIfExists` por `StringEquals`, porque el parámetro `ReturnValues` siempre tiene un valor (`NONE`, a no ser que se especifique otra cosa).
- Para permitir todos los atributos cuyo nombre coincida con un patrón, utilice `StringLike` en lugar de `StringEquals` y use un carácter comodín (*) que halle coincidencias de patrones con varios caracteres.

3: concesión de permisos para evitar actualizaciones en determinados atributos

La siguiente política de permisos limita el acceso del usuario de forma que únicamente pueda actualizar los atributos concretos que se identifican mediante la clave de condición `dynamodb:Attributes`. La condición `StringNotLike` impide que una aplicación actualice los atributos especificados con la clave de condición `dynamodb:Attributes`.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"PreventUpdatesOnCertainAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem"
         ],
         "Resource":"arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
         "Condition":{
            "ForAllValues:StringNotLike":{
               "dynamodb:Attributes":[
                  "FreeGamesAvailable",
                  "BossLevelUnlocked"
               ]
```

```
            },
            "StringEquals":{
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
            }
         }
      }
   ]
}
```

Tenga en cuenta lo siguiente:

- La acción `UpdateItem`, al igual que otras acciones de escritura, requiere acceso de lectura a los elementos para que pueda devolver valores antes y después de la actualización. En la política, se limita la acción para obtener acceso exclusivamente a los atributos que está permitido actualizar especificando la clave de condición `dynamodb:ReturnValues`. La clave de condición restringe el valor de `ReturnValues` en la solicitud porque solo permite especificar `NONE`, `UPDATED_OLD` o `UPDATED_NEW` y no incluye `ALL_OLD` ni `ALL_NEW`.
- Las acciones `PutItem` y `DeleteItem` sustituyen un elemento completo y, por consiguiente, permiten que las aplicaciones modifiquen cualquier atributo. Así pues, para limitar una aplicación de modo que únicamente pueda actualizar determinados atributos, no debe conceder permisos para estos API.

4: concesión de permisos para consultar únicamente los atributos proyectados de un índice

La siguiente política de permisos permite realizar consultas en un índice secundario (`TopScoreDateTimeIndex`) utilizando la clave de condición `dynamodb:Attributes`. Además, limita las consultas de tal forma que solo se puedan solicitar determinados atributos que se han proyectado en el índice.

Para exigir a la aplicación que especifique una lista de atributos en la consulta, la política especifica también la clave de condición `dynamodb:Select` de modo que requiere que el parámetro `Select` de la acción `Query` de DynamoDB sea `SPECIFIC_ATTRIBUTES`. La lista de atributos se limita a unos elementos concretos que se obtienen utilizando la clave de condición `dynamodb:Attributes`.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"QueryOnlyProjectedIndexAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:Attributes":[
                  "TopScoreDateTime",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "Attempts"
               ]
            },
            "StringEquals":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
         }
      }
   ]
}
```

La siguiente política de permisos es similar, pero la consulta debe solicitar todos los atributos que se han proyectado en el índice.

JSON

```
{
   "Version":"2012-10-17",
```

```
   "Statement":[
      {
         "Sid":"QueryAllIndexAttributes",
         "Effect":"Allow",
         "Action":[
            "dynamodb:Query"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "StringEquals":{
               "dynamodb:Select":"ALL_PROJECTED_ATTRIBUTES"
            }
         }
      }
   ]
}
```

5: concesión de permisos para limitar el acceso a determinados atributos y valores de la clave de partición

La política de permisos siguiente permite realizar acciones de DynamoDB concretas (que se han especificado en el componente `Action`) en una tabla y un índice de tabla (que se han especificado en el componente `Resource`). La política utiliza la clave de condición `dynamodb:LeadingKeys` para restringir los permisos de tal forma que únicamente incluyan los elementos cuyo valor de clave de partición coincida con el identificador de Facebook del usuario.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"LimitAccessToCertainAttributesAndKeyValues",
         "Effect":"Allow",
         "Action":[
            "dynamodb:UpdateItem",
            "dynamodb:GetItem",
            "dynamodb:Query",
```

```
            "dynamodb:BatchGetItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores",
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores/index/
TopScoreDateTimeIndex"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${graph.facebook.com:id}"
               ],
               "dynamodb:Attributes":[
                  "attribute-A",
                  "attribute-B"
               ]
            },
            "StringEqualsIfExists":{
               "dynamodb:Select":"SPECIFIC_ATTRIBUTES",
               "dynamodb:ReturnValues":[
                  "NONE",
                  "UPDATED_OLD",
                  "UPDATED_NEW"
               ]
            }
         }
      }
   ]
}
```

Tenga en cuenta lo siguiente:

- Las acciones de escritura permitidas por la política (`UpdateItem`) solo pueden modificar `attribute-A` o `attribute-B`.
- Dado que la política permite `UpdateItem`, una aplicación puede insertar nuevos elementos y los atributos ocultos serán null en los nuevos elementos. Si estos atributos se proyectan en `TopScoreDateTimeIndex`, la política incluirá la ventaja agregada de impedir las consultas que generen operaciones de recuperación (fetch) en la tabla.
- Las aplicaciones no pueden leer ningún atributo que no figure en `dynamodb:Attributes`. Con esta política en vigor, una aplicación debe establecer el parámetro `Select` en

`SPECIFIC_ATTRIBUTES` en las solicitudes de lectura y solamente se pueden solicitar los atributos que pertenecen a la lista blanca. En las solicitudes de escritura, la aplicación no puede establecer `ReturnValues` en `ALL_OLD` ni `ALL_NEW` ni puede llevar a cabo operaciones de escritura condicionales basadas en otros atributos que no sean estos.

Temas relacionados

- Identity and Access Management en Amazon DynamoDB
- Permisos de la API de DynamoDB: referencia acerca de acciones, recursos y condiciones

Uso de la federación de identidades web

Cuando escriba una aplicación dirigida a gran cantidad de usuarios, puede utilizar, si lo desea, identidades web federadas para llevar a cabo las operaciones de autenticación y autorización. Las identidades web federadas evitan tener que crear usuarios de individuales. En su lugar, los usuarios pueden iniciar sesión en un proveedor de identidades y obtener credenciales de seguridad temporales de AWS Security Token Service (AWS STS). A continuación, la aplicación puede utilizar estas credenciales para obtener acceso a los servicios de AWS.

Las identidades web federadas admiten los siguientes proveedores de identidad:

- Login with Amazon
- Facebook
- Google

Recursos adicionales para identidades web federadas

Los siguientes recursos pueden ayudarle a obtener más información sobre las identidades web federadas:

- El artículo Web Identity Federation using the AWS SDK para .NET (Identidad federada web usando NETlong) del blog de desarrolladores de AWS describe paso a paso cómo utilizar las identidades federadas web con Facebook. Incluye fragmentos de código en C# que muestran cómo asumir un rol de IAM con identidad web y cómo usar las credenciales de seguridad temporales para obtener acceso a un recurso de AWS.
- La AWS Mobile SDK for iOS y la AWS Mobile SDK for Android contienen aplicaciones de ejemplo. Incluyen código que muestra cómo invocar a los proveedores de identidad y, a continuación,

cómo utilizar la información de estos proveedores para recibir y utilizar credenciales de seguridad temporales.

- En el artículo Web Identity Federation with Mobile Applications se explican las identidades web federadas y se muestra un ejemplo de cómo utilizarlas para obtener acceso a un recurso de AWS.

Ejemplo de política para la federación de identidades web

Para mostrarle cómo utilizar las identidades federadas web con DynamoDB, vamos a retomar la tabla GameScores presentada en Uso de condiciones de las políticas de IAM para control de acceso preciso. Esta es la clave principal para GameScores.

Nombre de la tabla	Tipo de clave principal	Nombre y tipo de clave de partición	Nombre y tipo de clave de ordenación
GameScores (UserId, GameTitle, ...)	Compuesto	Nombre de atributo: UserId Tipo: cadena	Nombre de atributo: GameTitle Tipo: cadena

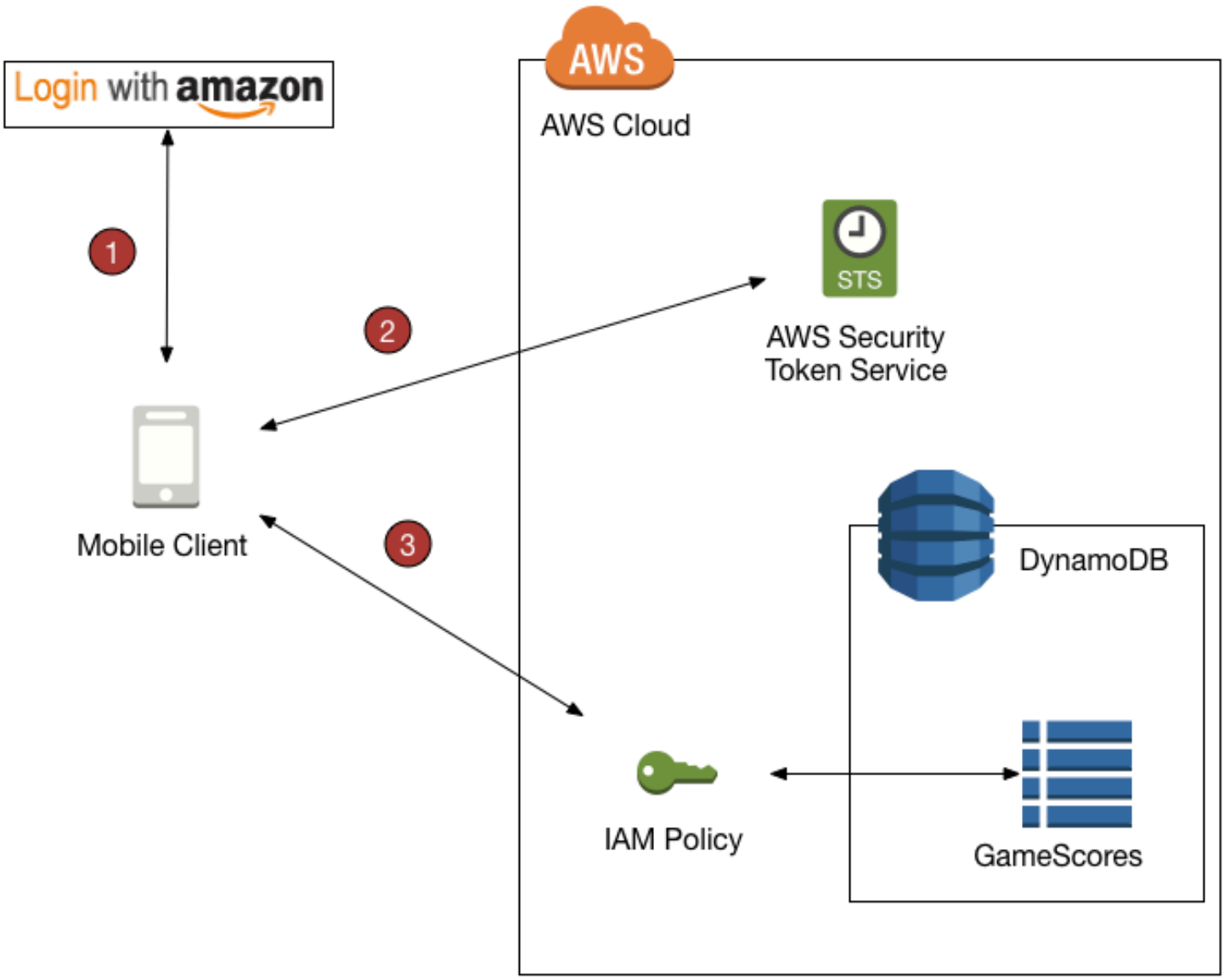
Ahora, supongamos que una aplicación de juegos para móviles utiliza la aplicación, con lo cual esta tiene que aceptar miles o incluso millones de usuarios. A esta escala, resulta muy complicado administrar individualmente a los usuarios de la aplicación y garantizar que cada uno de ellos únicamente pueda obtener acceso a sus propios datos en la tabla GameScores. Afortunadamente, muchos usuarios ya tiene cuentas de un proveedor de identidades tercero, como Facebook, Google o Login with Amazon. Por este motivo, es lógico utilizar alguno de ellos para llevar a cabo las tareas de autenticación.

Para hacer esto mediante las identidades web federadas, el desarrollador de la aplicación debe registrar esta última en un proveedor de identidades (como Login with Amazon) y obtener un identificador de aplicación exclusivo. A continuación, el desarrollador debe crear un rol de IAM. (En este ejemplo, este rol se llama GameRole). El rol debe tener adjunto un documento de política de IAM en el que se especifiquen las condiciones en virtud de las cuales la aplicación podrá obtener acceso a la tabla GameScores.

Cuando un usuario desee jugar, iniciará sesión en esta cuenta de Login with Amazon desde la propia aplicación de juegos. A continuación, la aplicación llama a AWS Security Token Service (AWS STS), lo que proporciona el ID de aplicación de Login with Amazon y solicita formar parte de GameRole.

AWS STS devolverá credenciales temporales de AWS a la aplicación y permitirá que esta última acceda a la tabla de GameScores, de acuerdo con lo establecido en el documento de las políticas de GameRole.

En el diagrama siguiente se muestra cómo se integran estas piezas entre sí.



Información general sobre la federación de identidades web

1. La aplicación llama a un proveedor de identidades tercero para autenticar al usuario y la aplicación. El proveedor de identidades devuelve un token de identidad web a la aplicación.
2. La aplicación llama a AWS STS y pasa el token de identidad web como información de entrada. AWS STS autoriza la aplicación y le proporciona credenciales temporales de acceso a AWS. Se

permite que la aplicación asuma un rol de IAM (GameRole) y que obtenga acceso a los recursos de AWS de conformidad con la política de seguridad del rol.

3. La aplicación llama a DynamoDB para acceder a la tabla GameScores. Dado que la aplicación ha asumido el rol GameRole, está sujeta a la política de seguridad asociada a ese rol. El documento de política impide que la aplicación obtenga acceso a datos que no pertenecen al usuario.

Una vez más, esta es la política de seguridad de GameRole que se mostró en Uso de condiciones de las políticas de IAM para control de acceso preciso:

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAccessToOnlyItemsMatchingUserID",
         "Effect":"Allow",
         "Action":[
            "dynamodb:GetItem",
            "dynamodb:BatchGetItem",
            "dynamodb:Query",
            "dynamodb:PutItem",
            "dynamodb:UpdateItem",
            "dynamodb:DeleteItem",
            "dynamodb:BatchWriteItem"
         ],
         "Resource":[
            "arn:aws:dynamodb:us-west-2:123456789012:table/GameScores"
         ],
         "Condition":{
            "ForAllValues:StringEquals":{
               "dynamodb:LeadingKeys":[
                  "${www.amazon.com:user_id}"
               ],
               "dynamodb:Attributes":[
                  "UserId",
                  "GameTitle",
                  "Wins",
                  "Losses",
                  "TopScore",
                  "TopScoreDateTime"
```

```
                ]
            },
            "StringEqualsIfExists":{
                "dynamodb:Select":"SPECIFIC_ATTRIBUTES"
            }
        }
    }
  ]
}
```

La cláusula `Condition` determina qué elementos de GameScores están visibles para la aplicación. Para ello, compara el identificador de Login with Amazon con los valores de las claves de partición `UserId` de `GameScores`. Solamente los elementos que pertenecen al usuario actual se pueden procesar utilizando una de las acciones de DynamoDB que se enumeran en esta política. No se puede obtener acceso a los demás elementos de la tabla. Además, solo se puede obtener acceso a los atributos específicos enumerados en la política.

Preparación para usar la federación de identidades web

Si es un desarrollador de aplicaciones y desea utilizar las identidades web federadas para la aplicación, siga estos pasos:

1. Inicie sesión como desarrollador con un proveedor de identidades tercero. Los siguientes enlaces externos proporcionan información sobre cómo inscribirse en los proveedores de identidad admitidos:
   - Registro con Amazon Developer Center
   - Registration en el sitio web de Facebook
   - Using OAuth 2.0 to Access Google APIs en el sitio de Google
2. Registre su aplicación en el proveedor de identidades. Cuando lo haga, el proveedor le proporcionará un identificador exclusivo de la aplicación. Si desea que la aplicación funcione con varios proveedores de identidades, tendrá que obtener un ID de aplicación de cada proveedor.
3. Cree uno o varios roles de IAM. Se requiere un rol para cada proveedor de identidades de cada aplicación. Por ejemplo, puede crear un rol que una aplicación asuma cuando el usuario inicie sesión mediante Login with Amazon, un segundo rol que la misma aplicación use cuando el usuario inicie sesión mediante Facebook y un tercer rol que dicha aplicación utilice cuando el usuario inicie sesión mediante Google.

Durante el proceso de creación de roles, deberá adjuntar una política de IAM al rol. El documento de política debe definir los recursos de DynamoDB que la aplicación requiere y los permisos para acceder a ellos.

Para obtener más información, consulte Información sobre la identidad federada web en la Guía del usuario de IAM.

> **Note**
>
> Como alternativa a AWS Security Token Service, puede utilizar Amazon Cognito. Amazon Cognito es el servicio preferido para administrar credenciales temporales de aplicaciones para móviles. Para obtener más información, consulte Obtención de credenciales en la Guía para desarrolladores de Amazon Cognito.

Generación de una política de IAM mediante la consola de DynamoDB

La consola de DynamoDB le puede ayudar a crear una política de IAM para utilizarla con las identidades federadas web. Para ello, se selecciona una tabla de DynamoDB y se especifica el proveedor de identidades, las acciones y los atributos que deben incluirse en la política. A continuación, la consola de DynamoDB genera una política que puede asociar a un rol de IAM.

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación, elija Tablas.
3. En la lista de tablas, elija aquella para la que desea crear la política de IAM.
4. Seleccione el botón Acciones y elija Crear directiva de control de acceso.
5. Elija el proveedor de identidades, las acciones y los atributos de la política.

   Cuando esté conforme con los ajustes, elija Generar política. Aparecerá la política generada.
6. Haga clic en Ver documentación y siga los pasos necesarios para adjuntar la política generada a un rol de IAM.

Cómo escribir la aplicación para usar la federación de identidades web

Para utilizar las identidades federadas web, la aplicación debe asumir el rol de IAM que ha creado. A partir de ese momento, la aplicación respetará la política de acceso que se haya asociado al rol.

En tiempo de ejecución, si la aplicación utiliza identidades web federadas, deberá seguir estos pasos:

1. Autenticarse ante el proveedor de identidades tercero. La aplicación debe llamar al proveedor de identidades mediante una interfaz proporcionada por este último. La forma exacta en la que se lleva a cabo la autenticación del usuario depende del proveedor y de la plataforma en la que se ejecute la aplicación. Por lo general, si el usuario aún no ha iniciado sesión, el proveedor de identidades (IdP) se encarga de mostrar una página de inicio de sesión propia.

   Una vez que el proveedor de identidades ha autenticado al usuario, devuelve un token de identidad web a la aplicación. El formato de este token depende del proveedor, pero suele ser una cadena muy larga de caracteres.
2. Obtenga credenciales temporales de seguridad de AWS. Para ello, la aplicación envía una solicitud `AssumeRoleWithWebIdentity` a AWS Security Token Service (AWS STS). Esta solicitud contiene los siguientes componentes:
   - El token de identidad web del paso anterior
   - El identificador de la aplicación del proveedor de identidades
   - El Nombre de recurso de Amazon (ARN) del rol de IAM que se ha creado para este proveedor de identidades y esta aplicación

   AWS STS devuelve un conjunto de credenciales de seguridad de AWS que vencen transcurrido un periodo determinado (el valor predeterminado son 3600 segundos).

   A continuación se muestra un ejemplo de una solicitud y la respuesta de una acción `AssumeRoleWithWebIdentity` en AWS STS. El token de identidad web se obtuvo del proveedor de identidades Login with Amazon.

```
GET / HTTP/1.1
Host: sts.amazonaws.com
Content-Type: application/json; charset=utf-8
URL: https://sts.amazonaws.com/?ProviderId=www.amazon.com
&DurationSeconds=900&Action=AssumeRoleWithWebIdentity
&Version=2011-06-15&RoleSessionName=web-identity-federation
&RoleArn=arn:aws:iam::123456789012:role/GameRole
```

```
&WebIdentityToken=Atza|IQEBLjAsAhQluyKqyBiYZ8-kclvGTYM81e...(remaining characters
 omitted)
```

```
<AssumeRoleWithWebIdentityResponse
  xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
  <AssumeRoleWithWebIdentityResult>
    <SubjectFromWebIdentityToken>amzn1.account.AGJZDKHJKAUUSW6C44CHPEXAMPLE</
SubjectFromWebIdentityToken>
    <Credentials>
      <SessionToken>AQoDYXdzEMf//////////wEa8AP6nNDwcSLnf+cHupC...(remaining
 characters omitted)</SessionToken>
      <SecretAccessKey>8Jhi60+EWUUbbUShTEsjTxqQtM8UKvsM6XAjdA==</SecretAccessKey>
      <Expiration>2013-10-01T22:14:35Z</Expiration>
      <AccessKeyId>06198791C436IEXAMPLE</AccessKeyId>
    </Credentials>
    <AssumedRoleUser>
      <Arn>arn:aws:sts::123456789012:assumed-role/GameRole/web-identity-federation</
Arn>
      <AssumedRoleId>AROAJU4SA2VW5SZRF2YMG:web-identity-federation</AssumedRoleId>
    </AssumedRoleUser>
  </AssumeRoleWithWebIdentityResult>
  <ResponseMetadata>
    <RequestId>c265ac8e-2ae4-11e3-8775-6969323a932d</RequestId>
  </ResponseMetadata>
</AssumeRoleWithWebIdentityResponse>
```

3. Acceder a recursos de AWS. La respuesta de AWS STS contiene información que la aplicación precisa para poder obtener acceso a los recursos de DynamoDB:

   - Los campos `AccessKeyID`, `SecretAccessKey` y `SessionToken` contienen credenciales de seguridad que son válidas para este usuario y esta aplicación exclusivamente.
   - El campo `Expiration` indica el límite de tiempo tras el cual estas credenciales ya no serán válidas.
   - El campo `AssumedRoleId` contiene el nombre de un rol de IAM que la aplicación ha asumido y es específico de la sesión. La aplicación respeta los controles de acceso del documento de política de IAM mientras dure esta sesión.
   - El campo `SubjectFromWebIdentityToken` contiene el ID exclusivo que aparece en una variable de política de IAM para este proveedor de identidades concreto. A continuación se muestran las variables de política de IAM correspondientes a los proveedores admitidos y algunos ejemplos de valores que pueden adoptar:

Variable de política	Ejemplo de valor
`${www.amazon.com:user_id}`	`amzn1.account.AGJZDKHJKAUUSW6C44CHPEXAMPLE`
`${graph.facebook.com:id}`	`123456789`
`${accounts.google.com:sub}`	`12345678901234567890 1`

Para obtener ejemplos de políticas de IAM en las que se utilizan estas variables de política, consulte Ejemplos de políticas: uso de condiciones para el control de acceso preciso.

Para obtener más información acerca de cómo se generan las credenciales temporales de acceso en AWS STS, consulte Solicitud de credenciales de seguridad temporales en la Guía del ususario de IAM.

Permisos de la API de DynamoDB: referencia acerca de acciones, recursos y condiciones

Cuando está configurando Identity and Access Management en Amazon DynamoDB y escribiendo una política de permisos que se pueda asociar a una identidad de IAM (políticas basadas en identidad), puede utilizar la lista de Claves de condición, recursos y acciones de Amazon DynamoDB en la Guía del usuario IAMcomo referencia. En la tabla siguiente, se muestran todas las operaciones de la API de DynamoDB, las acciones correspondientes a las que puede conceder permisos para realizar la acción y el recurso de AWS al que puede conceder los permisos. Las acciones se especifican en el campo `Action` de la política y el valor del recurso se especifica en el campo `Resource` de la política.

Puede utilizar claves de condiciones generales de AWS en sus políticas de DynamoDB para expresar condiciones. Para obtener una lista completa de claves amplias de AWS, consulte la Referencia de los elementos de política de IAM JSON en la Guía del usuario de IAM.

Además de las claves de condición que afectan a AWS en su conjunto, DynamoDB tiene sus propias claves que puede utilizar en las condiciones. Para obtener más información, consulte Uso de condiciones de las políticas de IAM para control de acceso preciso.

Temas relacionados

- Identity and Access Management en Amazon DynamoDB
- Uso de condiciones de las políticas de IAM para control de acceso preciso

Validación de conformidad por sector para DynamoDB

Para saber si un Servicio de AWS está incluido en el ámbito de programas de conformidad específicos, consulte Servicios de AWS en el ámbito del programa de conformidad y elija el programa de conformidad que le interese. Para obtener información general, consulte Programas de conformidad de AWS.

Puedes descargar los informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte Descarga de informes en AWS Artifact.

Su responsabilidad de conformidad al utilizar Servicios de AWS se determina en función de la confidencialidad de los datos, los objetivos de conformidad de su empresa, así como de la legislación y los reglamentos aplicables. Para obtener más información sobre su responsabilidad en cuanto a la conformidad al utilizar los Servicios de AWS, consulte Documentación de seguridad de AWS.

Resiliencia y recuperación de desastres en Amazon DynamoDB

La infraestructura global de AWS se compone de regiones y zonas de disponibilidad de AWS. AWS Las regiones proporcionan varias zonas de disponibilidad físicamente independientes y aisladas que se encuentran conectadas mediante redes con un alto nivel de rendimiento y redundancia, además de baja latencia. Con las zonas de disponibilidad, puede diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre zonas de disponibilidad sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de centros de datos únicos o múltiples.

Si necesita replicar sus datos o aplicaciones sobre grandes distancias geográficas, utilice las regiones locales de AWS. Una región local de AWS es un único centro de datos diseñado para complementar una región de AWS existente. Al igual que en el caso de todas las regiones de AWS, las regiones locales de AWS se encuentran completamente aisladas de las demás regiones de AWS.

Para obtener más información sobre las zonas de disponibilidad y las regiones de AWS, consulta Infraestructura global de AWS.

Además de la infraestructura global de AWS, Amazon DynamoDB ofrece varias características que lo ayudan con sus necesidades de resiliencia y backup de los datos.

Solicitud de copia de seguridad y restauración

DynamoDB proporciona una funcionalidad de backup en diferido. Le permite crear backup completos de las tablas para una retención y archivado a largo plazo. Para obtener más información, consulte Solicitud de copia de seguridad y restauración para DynamoDB.

Recuperación a un momento dado

La recuperación a un momento dado ayuda a proteger las tablas de DynamoDB de operaciones accidentales de escritura o eliminación. Al habilitar la recuperación a un momento dado, ya no hay que preocuparse por crear, mantener o planificar backups bajo demanda. Para obtener más información, consulte Recuperación a un momento dado en DynamoDB.

Tablas globales que se sincronizan entre regiones de AWS

DynamoDB distribuye automáticamente los datos y el tráfico de las tablas entre un número suficiente de servidores para satisfacer sus requisitos de almacenamiento y rendimiento, al mismo tiempo que mantiene un desempeño uniforme y rápido. Todos los datos se almacenan en discos de estado sólido (SSD) y se replican automáticamente en varias zonas de disponibilidad de una región de AWS, con objeto de ofrecer prestaciones integradas de alta disponibilidad y durabilidad de los datos. Puede utilizar tablas globales para mantener sincronizadas las tablas de DynamoDB en las regiones de AWS.

Seguridad de la infraestructura en Amazon DynamoDB

Como se trata de un servicio administrado, Amazon DynamoDB está protegido por los procedimientos de seguridad de red globales de AWS que se describen en la sección Protección de la infraestructura de AWS Well-Architected Framework.

Puede utilizar llamadas a la API publicadas de AWS para acceder a DynamoDB a través de la red. Los clientes pueden utilizar la versión 1.2 o 1.3. de TLS (seguridad de la capa de transporte). Los clientes también deben admitir conjuntos de cifrado con confidencialidad directa total (PFS) tales como Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos. Además, las solicitudes deben estar firmadas mediante un ID de clave de acceso y una clave de acceso secreta que esté asociada a una entidad principal de IAM. También puede utilizar AWS

Security Token Service (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

También puede utilizar un punto de enlace de la nube virtual privada (VPC) para DynamoDB para permitir que las instancias de Amazon EC2 de la VPC utilicen sus direcciones IP privadas para tener acceso a DynamoDB sin exponerse en la Internet pública. Para obtener más información, consulte Uso de puntos de conexión de Amazon VPC para tener acceso a DynamoDB.

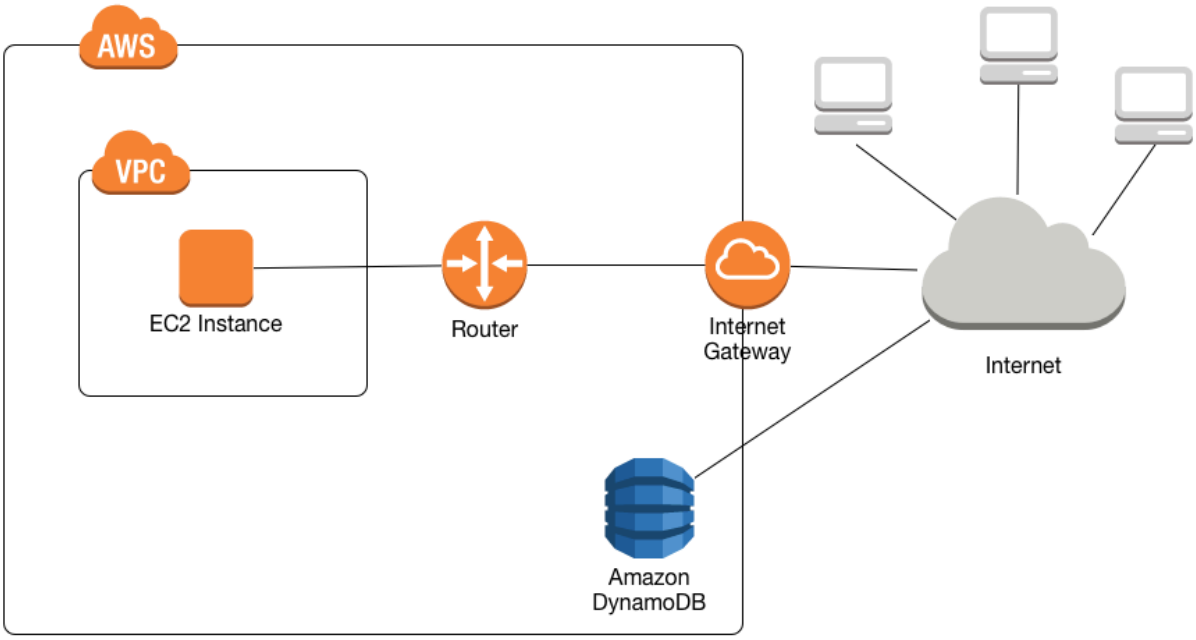
Uso de puntos de conexión de Amazon VPC para tener acceso a DynamoDB

Por razones de seguridad, muchos clientes de AWS ejecutan sus aplicaciones dentro de un entorno de Amazon Virtual Private Cloud (Amazon VPC). Con Amazon VPC, puede lanzar instancias de Amazon EC2 en una nube virtual privada que está aislada de forma lógica de otras redes, incluida la red pública de Internet. Con una Amazon VPC, puede controlar el rango de direcciones IP, las subredes, las tablas de enrutamiento, las gateways de red y los ajustes de seguridad.

> **Note**
>
> Si creó la Cuenta de AWS después del 4 de diciembre de 2013, ya dispone de una VPC predeterminada en cada Región de AWS. Las VPC predeterminadas están listas para usarse; puede comenzar de inmediato a usarlas sin tener que realizar ningún paso de configuración adicional.
> Para obtener más información acerca de las VPC predeterminadas, consulte VPC predeterminada y subredes predeterminadas en la Guía del usuario de Amazon VPC.

Para obtener acceso a la red pública de Internet, la VPC debe tener un gateway de Internet, es decir, un router virtual que conecta su VPC a Internet. Esto permite a las aplicaciones que se ejecutan en Amazon EC2 en su Amazon VPC acceder a los recursos de Internet, como Amazon DynamoDB.

De forma predeterminada, las comunicaciones de entrada y salida de DynamoDB usan el protocolo HTTPS, que protege el tráfico de la red mediante el uso del cifrado SSL/TLS. En el siguiente diagrama se muestra una instancia de Amazon EC2 en una VPC que accede a DynamoDB, por lo que DynamoDB utiliza una puerta de enlace de Internet en lugar de los puntos de conexión de VPC.



A muchos clientes les preocupa con razón la privacidad y la seguridad en el envío y recepción de datos a través de la red pública de Internet. Estas preocupaciones se pueden atajar con el uso de una red privada virtual (VPN) para dirigir todo el tráfico de la red de DynamoDB a través de su propia infraestructura de red corporativa. Sin embargo, este enfoque puede conllevar problemas de ancho de banda y disponibilidad.

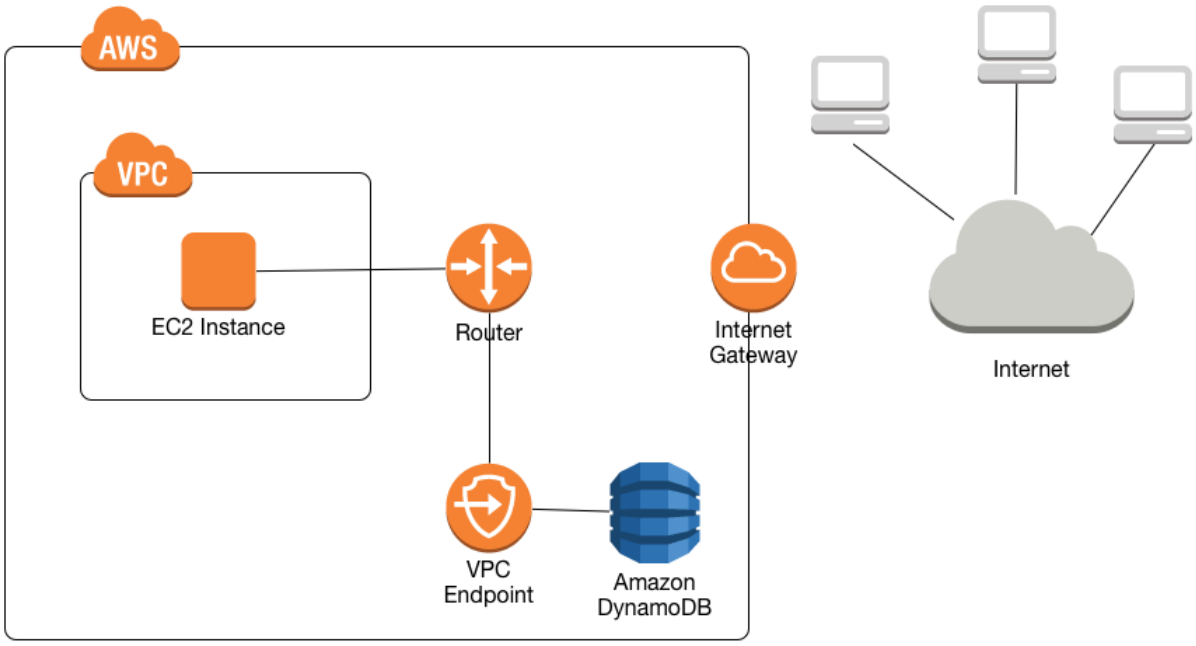
Estos problemas se pueden solventar con puntos de enlace de la VPC para DynamoDB. Un punto de enlace de la VPC para DynamoDB permite que las instancias de Amazon EC2 de la VPC utilicen sus direcciones IP privadas para acceder a DynamoDB sin exponerse en la Internet pública. Sus instancias EC2 no tienen que ser direcciones IP públicas, ni necesita un gateway de Internet, un dispositivo NAT o una gateway privada virtual en su VPC. Para controlar el acceso a DynamoDB se utilizan políticas de punto de enlace. El tráfico entre su VPC y el servicio AWS no sale de la red de Amazon.

> **Note**
>
> Al utilizar direcciones IP públicas, todas las comunicaciones de VPC entre instancias y servicios alojados en AWS se mantienen privadas dentro de la red de AWS. Los paquetes

que se originan en la red de AWS con un destino en la red de AWS permanecen en la red global de AWS, excepto el tráfico con destino u origen en las regiones de China de AWS.

Cuando crea un punto de enlace de la VPC para DynamoDB todas las solicitudes a un punto de enlace de DynamoDB dentro de la región (por ejemplo, dynamodb.us-west-2.amazonaws.com) se enrutan a un punto de enlace de DynamoDB privado dentro de la red de Amazon. No es necesario modificar las aplicaciones que se ejecutan en instancias EC2 en la VPC. El nombre del punto de enlace sigue siendo el mismo, pero la ruta a DynamoDB permanece por completo dentro de la red de Amazon y no accede a la red pública de Internet.

En el siguiente diagrama se muestra cómo una instancia EC2 de un punto de enlace de la VPC, puede utilizar un punto de conexión de la VPC para acceder a DynamoDB.



Para obtener más información, consulte the section called “Tutorial: Uso de un punto de conexión de VPC para DynamoDB”.

Compartir puntos de conexión de VPC de Amazon y DynamoDB

Para habilitar el acceso al servicio de DynamoDB a través del punto de conexión de una puerta de enlace de una subred de VPC, debe tener permisos de cuenta de propietario para esa subred de VPC.

Una vez que el punto de conexión de una puerta de enlace de una subred de VPC tenga acceso a DynamoDB, cualquier cuenta de AWS con acceso a esa subred puede usar DynamoDB. Esto significa que todos los usuarios de cuentas de la subred de VPC pueden usar cualquier tabla de DynamoDB a la que tengan acceso. Esto incluye las tablas de DynamoDB asociadas a una cuenta diferente a la de la subred de VPC. No obstante, el propietario de la subred de VPC puede restringir a cualquier usuario concreto de la subred el uso del servicio de DynamoDB a través del punto de conexión de la puerta de enlace, según su criterio.

Tutorial: Uso de un punto de conexión de VPC para DynamoDB

En esta sección se explica cómo configurar y usar un punto de enlace de la VPC para DynamoDB.

Temas

- Paso 1: lanzar una instancia de Amazon EC2
- Paso 2: configurar la instancia de Amazon EC2
- Paso 3: crear un punto de conexión de VPC para DynamoDB
- Paso 4: (Opcional) Eliminar

Paso 1: lanzar una instancia de Amazon EC2

En este paso, lanza una instancia de Amazon EC2 en su Amazon VPC predeterminada. A partir de ese momento, podrá crear y utilizar un punto de enlace de la VPC para DynamoDB.

1. Abra la consola de Amazon EC2 en https://console.aws.amazon.com/ec2/.
2. Elija Lanzar instancia y proceda del modo siguiente:

   Paso 1: Elegir una imagen de máquina de Amazon Machine Image (AMI)

   - En la parte superior de la lista de AMI, vaya a Amazon Linux AMI (AMI de Amazon Linux) y elija Select (Seleccionar).

   Paso 2: Elegir un tipo de instancia

- En la parte superior de la lista de tipos de instancias, elija t2.micro.
- Elija Next: Configure Instance Details.

Paso 3: Configurar los detalles de la instancia

- Vaya a Network (Red) y elija la VPC predeterminada.

  Elija Siguiente: Añadir almacenamiento.

Paso 4: Agregar almacenamiento

- Elija Next: Tag Instance para omitir este paso.

Paso 5: Etiquetar la instancia

- Elija Next: Configure Security Group para omitir este paso.

Paso 6: Configurar un grupo de seguridad

- Elija Seleccionar un grupo de seguridad existente.
- En la lista de grupos de seguridad, elija default. Se trata del grupo de seguridad predeterminado para la VPC.
- Elija Next: Review and Launch.

Paso 7: Revisar el lanzamiento de la instancia

- Elija Iniciar.

3. En la ventana Select an existing key pair or create a new key pair, proceda del modo siguiente:

   - Si no dispone de un par de claves de Amazon EC2, elija Create a new key pair (Crear un nuevo par de claves) y siga las instrucciones. Se le pedirá que descargue un archivo de clave privada (archivo .pem); lo necesitará más adelante cuando inicie sesión en la instancia de Amazon EC2.

- Si ya dispone de un par de claves de Amazon EC2, vaya a Select a key pair y elíjalo en la lista. Debe tener el archivo de clave privada (archivo .pem) a su disposición para poder iniciar sesión en la instancia de Amazon EC2.

4. Cuando haya configurado el par de claves, elija Launch Instances.
5. Vuelva a la página de inicio de la consola de Amazon EC2 y elija la instancia que ha lanzado. En el panel inferior, en la pestaña Description, busque el Public DNS de su instancia. Por ejemplo: `ec2-00-00-00-00.us-east-1.compute.amazonaws.com`.

   Anote el nombre de DNS público, pues lo necesitará en el paso siguiente de este tutorial (Paso 2: configurar la instancia de Amazon EC2).

> **Note**
>
> La instancia de Amazon EC2 tardará unos minutos en estar disponible. Antes de continuar con el siguiente paso, asegúrese de que Estado de la instancia tenga el valor `running` y de que se hayan superado todas las Comprobaciones de estado.

Paso 2: configurar la instancia de Amazon EC2

Una vez que la instancia de Amazon EC2 esté disponible, podrá iniciar sesión en ella y prepararla para utilizarla por primera vez.

> **Note**
>
> En los pasos siguientes se presupone que se va a conectar a la instancia de Amazon EC2 desde un ordenador que ejecuta Linux. Para obtener información sobre otras maneras de conectarse, consulte Conexión con la instancia de Linux en la Guía del usuario de Amazon EC2.

1. Tendrá que autorizar el tráfico SSH entrante en su instancia de Amazon EC2. Para ello, creará un nuevo grupo de seguridad de EC2 y, a continuación, asignará el grupo de seguridad a la instancia EC2.

   a. En el panel de navegación, elija Grupos de seguridad.

   b. Elija Crear grupo de seguridad. En la ventana Crear grupo de seguridad, haga lo siguiente:

- Security group name: escriba un nombre único para su grupo de seguridad. Por ejemplo: `my-ssh-access`
- Description: escriba una breve descripción del grupo de seguridad.
- VPC: seleccione la VPC predeterminada.
- En la sección Security group rules, elija Add Rule y proceda del modo siguiente:
  - Type: elija SSH.
  - Source: elija My IP.

Cuando esté conforme con los ajustes, elija Crear.

c. En el panel de navegación, seleccione Instances (Instancias).

d. Elija la instancia de Amazon EC2 que ha lanzado en Paso 1: lanzar una instancia de Amazon EC2.

e. Seleccione Actions --> Networking --> Change Security Groups.

f. En Cambiar Security Groups, seleccione el grupo de seguridad que ha creado anteriormente en este procedimiento (por ejemplo: `my-ssh-access`). El grupo de seguridad `default` existente también debe estar seleccionado. Cuando la configuración sea la que desea, elija Assign Security Groups.

2. Utilice el comando `ssh` para iniciar sesión en su instancia de Amazon EC2, como se muestra en el siguiente ejemplo.

```
ssh -i my-keypair.pem ec2-user@public-dns-name
```

Deberá especificar el archivo de clave privada (archivo .pem) y el nombre de DNS público de la instancia. (Consulte Paso 1: lanzar una instancia de Amazon EC2).

El identificador de inicio de sesión es `ec2-user`. No se requiere contraseña.

3. Configure las credenciales de AWS como se muestra en el siguiente ejemplo. Introduzca su ID de clave de acceso de AWS, una clave secreta, y el nombre de la región predeterminada cuando se le solicite.

```
aws configure
```

```
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

```
Default region name [None]: us-east-1
Default output format [None]:
```

Ahora está listo para crear un punto de enlace de la VPC para DynamoDB.

Paso 3: crear un punto de conexión de VPC para DynamoDB

En este paso, creará un punto de enlace de la VPC para DynamoDB y lo probará para asegurarse de que funciona.

1. Antes de empezar, compruebe que puede comunicarse con DynamoDB mediante su punto de enlace público.

   ```
   aws dynamodb list-tables
   ```

   La salida mostrará una lista de las tablas de DynamoDB que tiene actualmente. (Si no tiene tablas, la lista estará vacía).

2. Compruebe que DynamoDB es un servicio disponible para la creación de puntos de enlace de la VPC en la región de AWS. (El comando se muestra en negrita, seguido del resultado de ejemplo).

   ```
   aws ec2 describe-vpc-endpoint-services
   ```

   ```
   {
       "ServiceNames": [
           "com.amazonaws.us-east-1.s3",
           "com.amazonaws.us-east-1.dynamodb"
       ]
   }
   ```

   En el resultado de ejemplo, DynamoDB es uno de los servicios disponibles, por lo que puede a empezar crear un punto de enlace de la VPC para este servicio.

3. Determine el identificador de su VPC.

   ```
   aws ec2 describe-vpcs
   ```

   ```
   {
   ```

```
    "Vpcs": [
        {
            "VpcId": "vpc-0bbc736e",
            "InstanceTenancy": "default",
            "State": "available",
            "DhcpOptionsId": "dopt-8454b7e1",
            "CidrBlock": "172.31.0.0/16",
            "IsDefault": true
        }
    ]
}
```

En el resultado de ejemplo, el ID de la VPC es `vpc-0bbc736e`.

4. Crear el punto de enlace de la VPC. Para el parámetro `--vpc-id`, especifique el ID de la VPC del paso anterior. Utilice el parámetro `--route-table-ids` para asociar el punto de enlace con las tablas de enrutamiento.

```
aws ec2 create-vpc-endpoint --vpc-id vpc-0bbc736e --service-name com.amazonaws.us-
east-1.dynamodb --route-table-ids rtb-11aa22bb
```

```
{
    "VpcEndpoint": {
        "PolicyDocument": "{\"Version\":\"2008-10-17\",\"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"*\",\"Resource\":\"*\"}]}",
        "VpcId": "vpc-0bbc736e",
        "State": "available",
        "ServiceName": "com.amazonaws.us-east-1.dynamodb",
        "RouteTableIds": [
            "rtb-11aa22bb"
        ],
        "VpcEndpointId": "vpce-9b15e2f2",
        "CreationTimestamp": "2017-07-26T22:00:14Z"
    }
}
```

5. Compruebe que puede acceder a DynamoDB a través del punto de enlace de la VPC:

```
aws dynamodb list-tables
```

Si lo desea, puede probar otros comandos de la AWS CLI para DynamoDB. Para obtener más información, consulte Referencia de comandos de la AWS CLI.

Paso 4: (Opcional) Eliminar

Si desea eliminar los recursos que ha creado en este tutorial, siga estos procedimientos:

Para eliminar el punto de enlace de la VPC para DynamoDB

1. Inicie sesión en la instancia de Amazon EC2.
2. Determine el ID de punto de enlace de la VPC.

```
aws ec2 describe-vpc-endpoints
```

```
{
    "VpcEndpoint": {
        "PolicyDocument": "{\"Version\":\"2008-10-17\",\"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"*\",\"Resource\":\"*\"}]}",
        "VpcId": "vpc-0bbc736e",
        "State": "available",
        "ServiceName": "com.amazonaws.us-east-1.dynamodb",
        "RouteTableIds": [],
        "VpcEndpointId": "vpce-9b15e2f2",
        "CreationTimestamp": "2017-07-26T22:00:14Z"
    }
}
```

   En el resultado de ejemplo, el ID de punto de conexión de VPC es `vpce-9b15e2f2`.
3. Elimine el punto de conexión de VPC.

```
aws ec2 delete-vpc-endpoints --vpc-endpoint-ids vpce-9b15e2f2
```

```
{
    "Unsuccessful": []
}
```

   La matriz vacía `[]` indica que la operación se ha realizado correctamente (no hay solicitudes con error).

Para terminar la instancia de Amazon EC2

1. Abra la consola de Amazon EC2 en https://console.aws.amazon.com/ec2/.

2. En el panel de navegación, seleccione Instances (Instancias).
3. Seleccione la instancia de Amazon EC2.
4. Elija Actions (Acciones), Instance State (Estado de la instancia), Terminate (Terminar).
5. En la ventana de confirmación, elija Sí, terminar.

AWS PrivateLink para DynamoDB

Con AWS PrivateLink para DynamoDB, puede aprovisionar puntos de conexión de Amazon VPC de la interfaz (puntos de conexión de la interfaz) en su nube privada virtual (Amazon VPC). A estos puntos de conexión se puede acceder directamente desde las aplicaciones que se encuentran en las instalaciones a través de la VPN y Direct Connect, o bien, en una Región de AWS diferente mediante el emparejamiento de Amazon VPC. Al usar AWS PrivateLink y puntos de conexión de la interfaz, puede simplificar la conectividad de la red privada desde sus aplicaciones a DynamoDB.

Las aplicaciones de la VPC no necesitan direcciones IP públicas para comunicarse con DynamoDB mediante puntos de conexión de interfaz de VPC para operaciones de DynamoDB. Los puntos de conexión de la interfaz se representan mediante una o más interfaces de red elásticas (ENI) a las que se asignan direcciones IP privadas desde subredes de la Amazon VPC. Las solicitudes a DynamoDB a través de puntos de conexión de la interfaz permanecen en la red de Amazon. Asimismo, puede acceder a los puntos de conexión de la interfaz en su Amazon VPC desde aplicaciones en las instalaciones a través de AWS Direct Connect o AWS Virtual Private Network (Site-to-Site VPN). Para obtener más información sobre cómo conectar la Amazon VPC a la red en las instalaciones, consulte la Direct Connect User Guide y la AWS Site-to-Site VPN User Guide.

Para obtener información general sobre los puntos de conexión de interfaz, consulte Puntos de conexión de VPC de Amazon de interfaz (AWS PrivateLink) en la Guía de AWS PrivateLink. También se admite AWS PrivateLink para puntos de conexión de Amazon DynamoDB Streams. Para obtener más información, consulte the section called “AWS PrivateLink para DynamoDB Streams”.

Temas

- Tipos de puntos de conexión de Amazon VPC para Amazon DynamoDB
- Consideraciones sobre el uso de AWS PrivateLink para Amazon DynamoDB
- Creación de un punto de conexión de VPC de Amazon
- Acceso a los puntos de conexión de la interfaz de Amazon DynamoDB
- Acceso a tablas de DynamoDB y operaciones de la API de control desde los puntos de conexión de la interfaz de DynamoDB

- Actualización de una configuración DNS en las instalaciones
- Creación de una política de punto de conexión de Amazon VPC para DynamoDB
- Uso de puntos de conexión de DynamoDB con acceso privado de Consola de administración de AWS
- AWS PrivateLink para DynamoDB Streams
- Uso de AWS PrivateLink para DynamoDB Accelerator (DAX)

Tipos de puntos de conexión de Amazon VPC para Amazon DynamoDB

Puede utilizar dos tipos de puntos de conexión de Amazon VPC para acceder a Amazon DynamoDB: puntos de conexión de la puerta de enlace y puntos de conexión de la interfaz (mediante AWS PrivateLink). Un punto de conexión de la puerta de enlace es una puerta de enlace que se especifica en la tabla de enrutamiento para acceder a DynamoDB desde la Amazon VPC a través de la red de AWS. Los puntos de conexión de la interfaz amplían la funcionalidad de los puntos de conexión de la puerta de enlace al usar direcciones IP privadas para enviar solicitudes a DynamoDB desde la Amazon VPC, las instalaciones u otra Amazon VPC en otra Región de AWS mediante el emparejamiento de VPC o AWS Transit Gateway. Para obtener más información, consulte What is Amazon VPC peering? y Transit Gateway frente a emparejamiento de VPC.

Los puntos de enlace de la interfaz son compatibles con los puntos de enlace de gateway. Si tiene un punto de conexión de la puerta de enlace existente en la Amazon VPC, puede utilizar ambos tipos de puntos de conexión en la misma Amazon VPC.

Puntos de conexión de la puerta de enlace para DynamoDB	Puntos de conexión de la interfaz para DynamoDB
En ambos casos, el tráfico de red permanece en la red de AWS.	
Utilizar direcciones IP públicas de Amazon DynamoDB	Utilizar direcciones IP privadas de su Amazon VPC para acceder a Amazon DynamoDB
No permitir el acceso desde las instalaciones	Permitir el acceso desde las instalaciones
No permitir el acceso desde otra Región de AWS	Permitir el acceso desde un punto de conexión de Amazon VPC en otra Región de AWS mediante el uso de emparejamiento de VPC o AWS Transit Gateway

Puntos de conexión de la puerta de enlace para DynamoDB	Puntos de conexión de la interfaz para DynamoDB
No facturado	Facturado

Para obtener más información acerca de los puntos de conexión de la puerta de enlace, consulte Puntos de conexión de Amazon VPC de la puerta de enlace en la Guía del usuario de AWS PrivateLink.

Consideraciones sobre el uso de AWS PrivateLink para Amazon DynamoDB

Las consideraciones sobre Amazon VPC se aplican a AWS PrivateLink para Amazon DynamoDB. Para obtener más información, consulte Consideraciones de los puntos de conexión de la interfaz y Cuotas de AWS PrivateLink en la Guía de AWS PrivateLink. Además, se aplican las siguientes restricciones.

AWS PrivateLink para Amazon DynamoDB no admite lo siguiente:

- Seguridad de la capa de transporte (TLS) 1.1
- Servicios de sistema de nombres de dominio (DNS) privado e híbrido

Puede enviar hasta 50 000 solicitudes por segundo para cada punto de conexión de AWS PrivateLink que active.

> **Note**
>
> Los tiempos de espera de conectividad de red con los puntos de conexión de AWS PrivateLink no están incluidos en el ámbito de las respuestas de error de DynamoDB y las aplicaciones que se conecten a los puntos de conexión de PrivateLink deberán gestionarlos adecuadamente.

Creación de un punto de conexión de VPC de Amazon

Para crear un punto de conexión de la interfaz de Amazon VPC, consulte Create an Amazon VPC endpoint en la Guía de AWS PrivateLink.

Acceso a los puntos de conexión de la interfaz de Amazon DynamoDB

Cuando se crea un punto de conexión de la interfaz, DynamoDB genera dos tipos de nombres DNS de DynamoDB específicos del punto de conexión: regional y zonal.

- Un nombre DNS regional incluye un ID único de punto de conexión de Amazon VPC, un identificador de servicio, la Región de AWS y `vpce.amazonaws.com` en el nombre. Por ejemplo, para el ID de punto de conexión de Amazon VPC *`vpce-1a2b3c4d`*, el nombre DNS generado podría ser similar a *`vpce-1a2b3c4d-5e6f`*`.dynamodb.us-east-1.vpce.amazonaws.com`.
- Un nombre de DNS zonal incluye la zona de disponibilidad, por ejemplo, *`vpce-1a2b3c4d-5e6f`***`-us-east-1a.`**`dynamodb.us-east-1.vpce.amazonaws.com`. Puede utilizar esta opción si la arquitectura aísla Zonas de disponibilidad. Por ejemplo, podría usarlo para la contención de fallos o para reducir los costos de transferencia de datos regionales.

> **Note**
>
> Para lograr una fiabilidad óptima, recomendamos implementar el servicio en un mínimo de tres zonas de disponibilidad.

Acceso a tablas de DynamoDB y operaciones de la API de control desde los puntos de conexión de la interfaz de DynamoDB

Puede usar la AWS CLI o los SDK de AWS para acceder a las tablas de DynamoDB y controlar las operaciones de la API a través de los puntos de conexión de la interfaz de DynamoDB.

AWS CLIEjemplos de

Para acceder a las tablas de DynamoDB o a las operaciones de la API de control de DynamoDB a través de los puntos de conexión de la interfaz de DynamoDB en los comandos AWS CLI, use los parámetros `--region` y `--endpoint-url`.

Ejemplo: crear un punto de conexión de VPC

```
aws ec2 create-vpc-endpoint \
--region us-east-1 \
--service-name com.amazonaws.us-east-1.dynamodb \
--vpc-id client-vpc-id \
```

```
--subnet-ids client-subnet-id \
--vpc-endpoint-type Interface \
--security-group-ids client-sg-id
```

Ejemplo: Modificar un punto de conexión de VPC

```
aws ec2 modify-vpc-endpoint \
--region us-east-1 \
--vpc-endpoint-id client-vpc-endpoint-id \
--policy-document policy-document \ #example optional parameter
--add-security-group-ids security-group-ids \ #example optional parameter
# any additional parameters needed, see Privatelink documentation for more details
```

Ejemplo: Enumerar las tablas mediante una URL de punto de conexión

En el siguiente ejemplo, reemplace la región `us-east-1` y el nombre DNS del ID de punto de conexión de VPC `vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` con su información.

```
aws dynamodb --region us-east-1 --endpoint https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com list-tables
```

Ejemplos del SDK de AWS

Para acceder a las tablas de DynamoDB o a las operaciones de la API de control de DynamoDB a través de los puntos de conexión de la interfaz de DynamoDB al utilizar los SDK de AWS, actualice sus SDK a la versión actual. A continuación, configure los clientes para que utilicen una URL de punto de conexión para acceder a una tabla o a una operación de la API de control de DynamoDB a través de los puntos de conexión de la interfaz de DynamoDB.

SDK for Python (Boto3)

Ejemplo: Utilizar una URL de punto de conexión para acceder a una tabla de DynamoDB

En el siguiente ejemplo, reemplace la región `us-east-1` y el ID de punto de conexión de VPC `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` con su información.

```
ddb_client = session.client(
service_name='dynamodb',
region_name='us-east-1',
```

```
endpoint_url='https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com'
)
```

SDK for Java 1.x

Ejemplo: Utilizar una URL de punto de conexión para acceder a una tabla de DynamoDB

En el siguiente ejemplo, reemplace la región `us-east-1` y el ID de punto de conexión de VPC `https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` con su información.

```
//client build with endpoint config
final AmazonDynamoDB dynamodb =
 AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(
        new AwsClientBuilder.EndpointConfiguration(
                "https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com",
                Regions.DEFAULT_REGION.getName()
        )
).build();
```

SDK for Java 2.x

Ejemplo: Uso de una URL de punto de conexión para acceder a una tabla de DynamoDB

En el siguiente ejemplo, reemplace la región us-east-1 y el ID de punto de conexión de VPC https://vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com por su propia información.

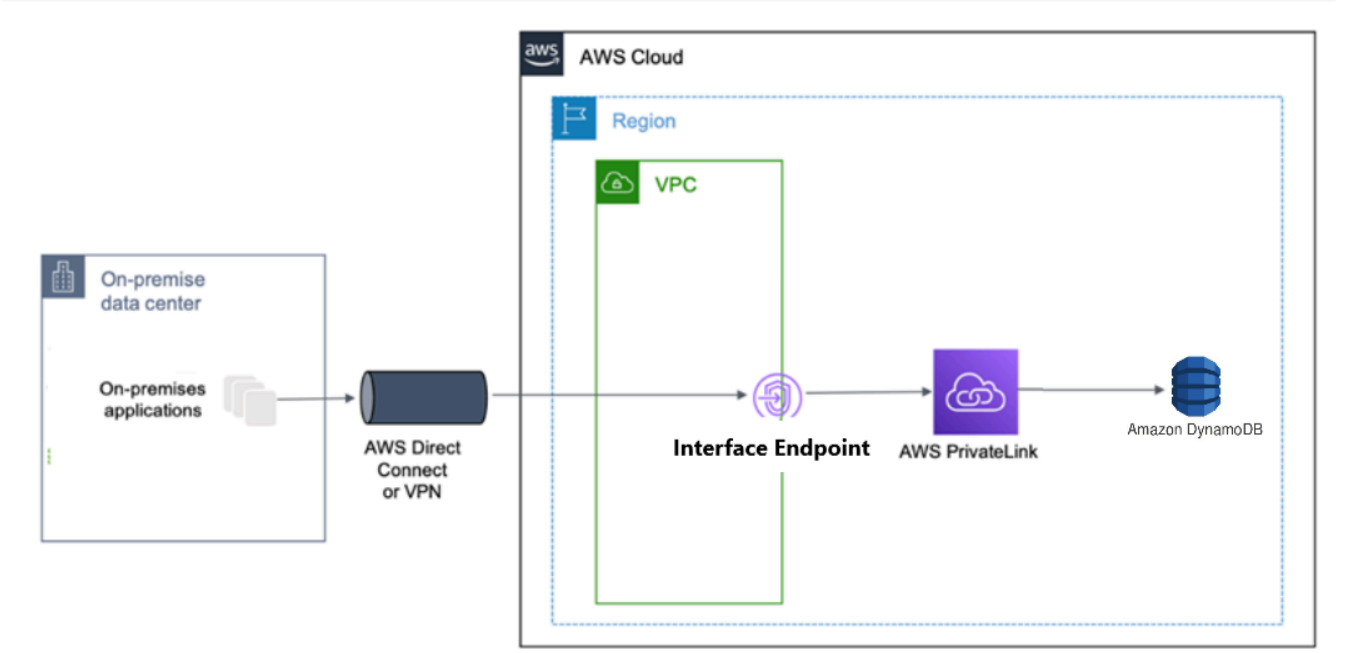
```
Region region = Region.US_EAST_1;
dynamoDbClient = DynamoDbClient.builder().region(region)
.endpointOverride(URI.create("https://vpce-1a2b3c4d-5e6f.dynamodb.us-
east-1.vpce.amazonaws.com"))
.build()
```

Actualización de una configuración DNS en las instalaciones

Al utilizar nombres DNS específicos de puntos de conexión para acceder a los puntos de conexión de la interfaz de DynamoDB, no es necesario actualizar la resolución DNS en las instalaciones. Puede resolver el nombre DNS específico del punto de conexión con la dirección IP privada del punto de conexión de la interfaz desde el dominio DNS público de DynamoDB.

Uso de puntos de conexión de la interfaz para acceder a DynamoDB sin un punto de conexión de la puerta de enlace o una puerta de enlace de Internet en la Amazon VPC

Los puntos de conexión de la interfaz de su Amazon VPC pueden dirigir tanto las aplicaciones en Amazon VPC como las aplicaciones en las instalaciones hacia DynamoDB a través de la red de Amazon, tal como se muestra en el siguiente diagrama.

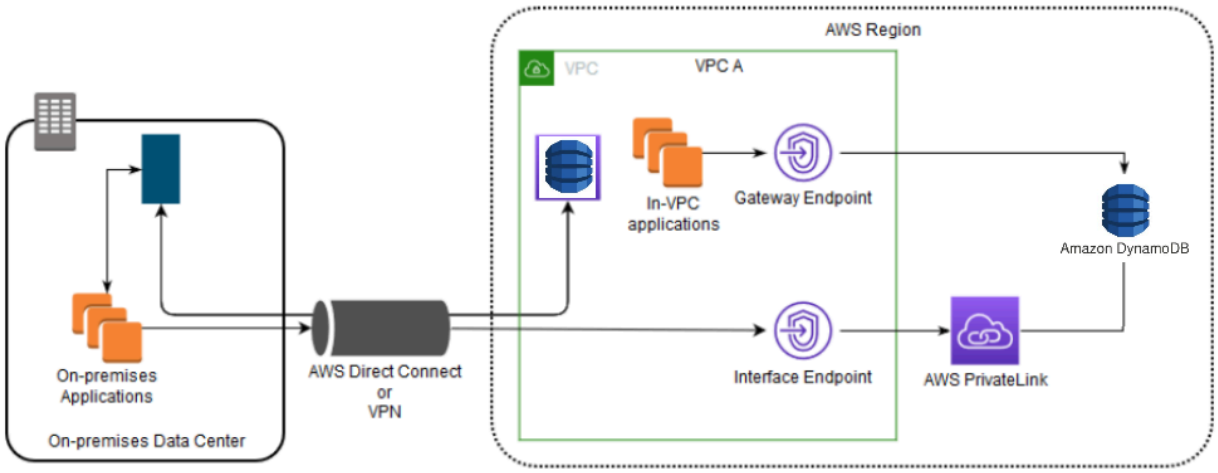


En el siguiente diagrama se ilustra lo siguiente:

- Su red en las instalaciones utiliza Direct Connect o Site-to-Site VPN para conectarse a la Amazon VPC A.
- Las aplicaciones en las instalaciones y en la Amazon VPC A utilizan nombres DNS específicos del punto de conexión para acceder a DynamoDB a través del punto de conexión de la interfaz de DynamoDB.
- Las aplicaciones en las instalaciones envían datos al punto de conexión de la interfaz en la Amazon VPC a través de Direct Connect (o Site-to-Site VPN). AWS PrivateLink transfiere los datos desde el punto de conexión de la interfaz hasta DynamoDB a través de la red de AWS.
- Las aplicaciones en la Amazon VPC también envían tráfico al punto de conexión de la interfaz. AWS PrivateLink transfiere los datos desde el punto de conexión de la interfaz a DynamoDB a través de la red de AWS.

Uso de puntos de conexión de la puerta de enlace y puntos de conexión de la interfaz juntos en la misma Amazon VPC para acceder a DynamoDB

Puede crear puntos de conexión de la interfaz y conservar el punto de conexión de la puerta de enlace existente en la misma Amazon VPC, tal como se muestra en el siguiente diagrama. De este modo, permite que las aplicaciones en la Amazon VPC continúen accediendo a DynamoDB a través del punto de conexión de la puerta de enlace, que no se factura. En ese caso, solo las aplicaciones en las instalaciones utilizarían puntos de conexión de la interfaz para acceder a DynamoDB. Para acceder a DynamoDB de esta manera, debe actualizar las aplicaciones en las instalaciones para que utilicen nombres DNS específicos de puntos de conexión para DynamoDB.



En el siguiente diagrama se ilustra lo siguiente:

- Las aplicaciones en las instalaciones utilizan nombres de DNS específicos de cada punto de conexión para enviar datos al punto de conexión de la interfaz dentro de la Amazon VPC a través de Direct Connect (o Site-to-Site VPN). AWS PrivateLink transfiere los datos desde el punto de conexión de la interfaz hasta DynamoDB a través de la red de AWS.
- Mediante el uso de nombres regionales predeterminados de DynamoDB, las aplicaciones en la Amazon VPC envían datos al punto de conexión de la puerta de enlace que se conecta a DynamoDB a través de la red de AWS.

Para obtener más información acerca de los puntos de conexión de la puerta de enlace, consulte Gateway Amazon VPC endpoints en la Guía del usuario de Amazon VPC.

Creación de una política de punto de conexión de Amazon VPC para DynamoDB

Puede asociar una política de punto de conexión con el punto de conexión de Amazon VPC que controla el acceso a DynamoDB. La política especifica la siguiente información:

- La entidad principal de AWS Identity and Access Management (IAM) que puede realizar acciones
- Las acciones que se pueden realizar
- Los recursos en los que se pueden llevar a cabo las acciones

Temas

- Ejemplo: Restringir el acceso a una tabla específica desde un punto de conexión de Amazon VPC

Ejemplo: Restringir el acceso a una tabla específica desde un punto de conexión de Amazon VPC

Puede crear una política de punto de conexión que restrinja el acceso a tablas específicas de DynamoDB. Este tipo de política es útil si tiene otros Servicios de AWS en su Amazon VPC que utilicen tablas. La siguiente política de tablas restringe el acceso únicamente a la *DOC-EXAMPLE-TABLE*. Para utilizar esta política de puntos de conexión, sustituya *DOC-EXAMPLE-TABLE* por el nombre de su tabla.

JSON

```
{
"Version":"2012-10-17",
  "Id": "Policy1216114807515",
  "Statement": [
    { "Sid": "Access-to-specific-table-only",
      "Principal": "*",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:dynamodb:us-east-1:111122223333:table/DOC-EXAMPLE-
TABLE",
```

```
                "arn:aws:dynamodb:us-east-1:111122223333:table/DOC-EXAMPLE-
TABLE/*"]
    }
  ]
}
```

Uso de puntos de conexión de DynamoDB con acceso privado de Consola de administración de AWS

Debe configurar el DNS para DynamoDB y DynamoDB Streams cuando utilice puntos de conexión de VPC con la consola de DynamoDB en Consola de administración de AWS Private Access.

Para configurar que DynamoDB sea accesible en Consola de administración de AWS Private Access, debe crear los dos puntos de conexión de VPC siguientes:

- `com.amazonaws.<region>.dynamodb`
- `com.amazonaws.<region>.dynamodb-streams`

Al crear los puntos de conexión de VPC, vaya a la consola de Route53 y cree una zona alojada privada para DynamoDB mediante el punto de conexión regional `dynamodb.us-east-1.amazonaws.com`.

Cree los siguientes dos registros de alias en la zona alojada privada:

- `dynamodb.<region>.amazonaws.com` que enruta el tráfico al punto de conexión de VPC `com.amazonaws.<region>.dynamodb`.
- `streams.dynamodb.<region>.amazonaws.com` que enruta el tráfico al punto de conexión de VPC `com.amazonaws.<region>.dynamodb-streams`.

AWS PrivateLink para DynamoDB Streams

Con AWS PrivateLink para Amazon DynamoDB Streams, puede aprovisionar puntos de conexión de Amazon VPC de la interfaz (puntos de conexión de la interfaz) en la nube privada virtual (Amazon VPC). A estos puntos de conexión se puede acceder directamente desde las aplicaciones que se encuentran en las instalaciones a través de la VPN y Direct Connect, o bien, en una Región de AWS diferente mediante el emparejamiento de Amazon VPC. Al usar AWS PrivateLink y puntos de

conexión de la interfaz, puede simplificar la conectividad de la red privada desde las aplicaciones a DynamoDB Streams.

Las aplicaciones de la Amazon VPC no necesitan direcciones IP públicas para comunicarse con DynamoDB Streams mediante puntos de conexión de interfaz de Amazon VPC para operaciones de DynamoDB Streams. Los puntos de conexión de la interfaz se representan mediante una o más interfaces de red elásticas (ENI) a las que se asignan direcciones IP privadas desde subredes de la Amazon VPC. Las solicitudes a DynamoDB Streams a través de puntos de conexión de interfaz permanecen en la red de Amazon. Asimismo, puede acceder a los puntos de conexión de la interfaz en la Amazon VPC desde aplicaciones en las instalaciones a través de Direct Connect o AWS Virtual Private Network (AWS VPN). Para obtener más información sobre cómo conectar la AWS Virtual Private Network a la red en las instalaciones, consulte la Guía del usuario de Direct Connect y la Guía del usuario de AWS Site-to-Site VPN.

Para obtener información general sobre los puntos de conexión de interfaz, consulte Puntos de conexión de Amazon VPC de interfaz (AWS PrivateLink).

> **Note**
>
> Solo se admiten puntos de conexión de interfaz en DynamoDB Streams. Los puntos de conexión de puerta de enlace no se admiten.

Temas

- Consideraciones sobre el uso de AWS PrivateLink para Amazon DynamoDB Streams
- Creación de un punto de conexión de VPC de Amazon
- Acceso a los puntos de conexión de interfaz de Amazon DynamoDB Streams
- Acceso a operaciones de la API de DynamoDB Streams desde los puntos de conexión de interfaz de DynamoDB Streams
- Ejemplos del SDK de AWS
- Creación de una política de punto de conexión de Amazon VPC para DynamoDB Streams
- Uso de puntos de conexión de DynamoDB con acceso privado de Consola de administración de AWS

Consideraciones sobre el uso de AWS PrivateLink para Amazon DynamoDB Streams

Las consideraciones sobre Amazon VPC se aplican a AWS PrivateLink para Amazon DynamoDB Streams. Para obtener más información, consulte Consideraciones de los puntos de conexión de interfaz y Cuotas de AWS PrivateLink. Se aplican las siguientes restricciones.

AWS PrivateLink para Amazon DynamoDB Streams no admite lo siguiente:

- Seguridad de la capa de transporte (TLS) 1.1
- Servicios de sistema de nombres de dominio (DNS) privado e híbrido

> **Note**
>
> Los tiempos de espera de conectividad de red con los puntos de conexión de AWS PrivateLink no están incluidos en el ámbito de las respuestas de error de DynamoDB y las aplicaciones que se conecten a los puntos de conexión de AWS PrivateLink deberán gestionarlos adecuadamente.

Creación de un punto de conexión de VPC de Amazon

Para crear un punto de conexión de la interfaz de Amazon VPC, consulte Create an Amazon VPC endpoint en la Guía de AWS PrivateLink.

Acceso a los puntos de conexión de interfaz de Amazon DynamoDB Streams

Cuando crea un punto de conexión de interfaz, DynamoDB genera dos tipos de nombres DNS de DynamoDB Streams específicos del punto de conexión: regional y zonal.

- Un nombre DNS regional incluye un ID único de punto de conexión de Amazon VPC, un identificador de servicio, la Región de AWS y `vpce.amazonaws.com` en el nombre. Por ejemplo, para el ID de punto de conexión de Amazon VPC *`vpce-1a2b3c4d`*, el nombre DNS generado podría ser similar a *`vpce-1a2b3c4d-5e6f`*`.streams.dynamodb.us-east-1.vpce.amazonaws.com`.
- Un nombre de DNS zonal incluye la zona de disponibilidad, por ejemplo, *`vpce-1a2b3c4d-5e6f`*`-us-east-1a.streams.dynamodb.us-east-1.vpce.amazonaws.com`. Puede utilizar esta opción si la arquitectura aísla Zonas de

disponibilidad. Por ejemplo, podría usarlo para la contención de fallos o para reducir los costos de transferencia de datos regionales.

Acceso a operaciones de la API de DynamoDB Streams desde los puntos de conexión de interfaz de DynamoDB Streams

Puede utilizar la AWS CLI o los AWS SDK para acceder a operaciones de API de DynamoDB Streams a través de puntos de conexión de interfaz de DynamoDB Streams.

AWS CLIEjemplos de

Para acceder a operaciones de DynamoDB Streams o de la API a través de puntos de conexión de interfaz de DynamoDB Streams en comandos de la AWS CLI, utilice los parámetros `--region` y `--endpoint-url`.

Ejemplo: crear un punto de conexión de VPC

```
aws ec2 create-vpc-endpoint \
--region us-east-1 \
--service-name com.amazonaws.us-east-1.dynamodb-streams \
--vpc-id client-vpc-id \
--subnet-ids client-subnet-id \
--vpc-endpoint-type Interface \
--security-group-ids client-sg-id
```

Ejemplo: Modificar un punto de conexión de VPC

```
aws ec2 modify-vpc-endpoint \
--region us-east-1 \
--vpc-endpoint-id client-vpc-endpoint-id \
--policy-document policy-document \ #example optional parameter
--add-security-group-ids security-group-ids \ #example optional parameter
# any additional parameters needed, see Privatelink documentation for more details
```

Ejemplo: enumerar flujos con una URL de punto de conexión

En el siguiente ejemplo, reemplace la región `us-east-1` y el nombre DNS del ID de punto de conexión de VPC `vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` con su información.

```
aws dynamodbstreams --region us-east-1 –endpoint https://
vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com list-streams
```

Ejemplos del SDK de AWS

Para acceder a las operaciones de la API de Amazon DynamoDB Streams a través de los puntos de conexión de interfaz de DynamoDB Streams al utilizar los AWS SDK, actualice los SDK a la versión actual. A continuación, configure los clientes para que utilicen una URL de punto de conexión para la API de control de DynamoDB Streams a través de los puntos de conexión de la interfaz de DynamoDB Streams.

SDK for Python (Boto3)

Ejemplo: utilizar una URL de punto de conexión para acceder a un flujo de DynamoDB

En el siguiente ejemplo, reemplace la región `us-east-1` y el ID de punto de conexión de VPC `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` con su información.

```
ddb_streams_client = session.client(
service_name='dynamodbstreams',
region_name='us-east-1',
endpoint_url='https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-
east-1.vpce.amazonaws.com'
)
```

SDK for Java 1.x

Ejemplo: utilizar una URL de punto de conexión para acceder a un flujo de DynamoDB

En el siguiente ejemplo, reemplace la región `us-east-1` y el ID de punto de conexión de VPC `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` con su información.

```
//client build with endpoint config
final AmazonDynamoDBStreams dynamodbstreams =
 AmazonDynamoDBStreamsClientBuilder.standard().withEndpointConfiguration(
        new AwsClientBuilder.EndpointConfiguration(
                "https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-
east-1.vpce.amazonaws.com",
```

```
                Regions.DEFAULT_REGION.getName()
        )
).build();
```

SDK for Java 2.x

Ejemplo: utilizar una URL de punto de conexión para acceder a un flujo de DynamoDB

En el siguiente ejemplo, reemplace la región `us-east-1` y el ID de punto de conexión de VPC `https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-east-1.vpce.amazonaws.com` con su información.

```
Region region = Region.US_EAST_1;
dynamoDbStreamsClient = DynamoDbStreamsClient.builder().region(region)
.endpointOverride(URI.create("https://vpce-1a2b3c4d-5e6f.streams.dynamodb.us-
east-1.vpce.amazonaws.com"))
.build()
```

Creación de una política de punto de conexión de Amazon VPC para DynamoDB Streams

Puede asociar una política de punto de conexión con el punto de conexión de Amazon VPC que controla el acceso a DynamoDB Streams. La política especifica la siguiente información:

- La entidad principal de AWS Identity and Access Management (IAM) que puede realizar acciones
- Las acciones que se pueden realizar
- Los recursos en los que se pueden llevar a cabo las acciones

Temas

- Ejemplo: restringir el acceso a un flujo específico desde un punto de conexión de Amazon VPC

Ejemplo: restringir el acceso a un flujo específico desde un punto de conexión de Amazon VPC

Puede crear una política de punto de conexión que restrinja el acceso a flujos de DynamoDB Streams específicos. Este tipo de política es útil si tiene otros Servicios de AWS en la Amazon VPC que utilicen DynamoDB Streams. La siguiente política de flujo restringe el acceso solo al flujo *2025-02-20T11:22:33.444* adjuntado a *DOC-EXAMPLE-TABLE*. Para utilizar esta

política de puntos de conexión, reemplace *DOC-EXAMPLE-TABLE* por el nombre de la tabla y *2025-02-20T11:22:33.444* por la etiqueta de flujo.

JSON

```
{
"Version":"2012-10-17",
  "Id": "Policy1216114807515",
  "Statement": [
    { "Sid": "Access-to-specific-stream-only",
      "Principal": "*",
      "Action": [
        "dynamodb:DescribeStream",
        "dynamodb:GetRecords"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:dynamodb:us-east-1:111122223333:table/table-name/
stream/2025-02-20T11:22:33.444"]
    }
  ]
}
```

> **Note**
>
> Los puntos de conexión de puerta de enlace no son compatibles con DynamoDB Streams.

Uso de puntos de conexión de DynamoDB con acceso privado de Consola de administración de AWS

Debe configurar el DNS para DynamoDB y DynamoDB Streams cuando utilice puntos de conexión de VPC con la consola de DynamoDB en Consola de administración de AWS Private Access.

Para configurar que DynamoDB sea accesible en Consola de administración de AWS Private Access, debe crear los dos puntos de conexión de VPC siguientes:

- `com.amazonaws.<region>.dynamodb`
- `com.amazonaws.<region>.dynamodb-streams`

Al crear los puntos de conexión de VPC, vaya a la consola de Route53 y cree una zona alojada privada para DynamoDB mediante el punto de conexión regional `dynamodb.us-east-1.amazonaws.com`.

Cree los siguientes dos registros de alias en la zona alojada privada:

- `dynamodb.<region>.amazonaws.com` que enruta el tráfico al punto de conexión de VPC `com.amazonaws.<region>.dynamodb`.
- `streams.dynamodb.<region>.amazonaws.com` que enruta el tráfico al punto de conexión de VPC `com.amazonaws.<region>.dynamodb-streams`.

Uso de AWS PrivateLink para DynamoDB Accelerator (DAX)

AWS PrivateLink para DynamoDB Accelerator (DAX) le permite acceder de forma segura a las API de administración de DAX, como `CreateCluster`, `DescribeClusters` y `DeleteCluster` y a través de direcciones IP privadas, dentro de la nube privada virtual (VPC). Esta característica le permite acceder a los servicios de DAX de forma privada desde las aplicaciones sin exponer el tráfico al Internet público.

DAX PrivateLink admite puntos de conexión de doble pila (`dax.{region}.api.aws`), lo que permite la conectividad IPv4 e IPv6. Con AWS PrivateLink para DAX, los clientes pueden acceder al servicio mediante nombres DNS privados. El soporte para puntos de conexión de doble pila garantiza una conectividad transparente y, al mismo tiempo, mantiene la privacidad de la red. Esto le permite acceder a DAX a través de puntos de conexión públicos de Internet y de VPC sin realizar ningún cambio en la configuración del SDK.

Consideraciones al usar AWS PrivateLink para DynamoDB Accelerator (DAX)

Al realizar la implementación AWS PrivateLink para DynamoDB Accelerator (DAX), se deben tener en cuenta varias consideraciones importantes.

Antes de configurar un punto de conexión de interfaz para DAX, tenga en cuenta lo siguiente:

- Los puntos de conexión de la interfaz de DAX solo admiten el acceso a las API de administración de DAX dentro de la misma Región de AWS. No puede usar un punto de conexión de interfaz para acceder a las API de administración de DAX en otras regiones.
- Para acceder a la Consola de administración de AWS de forma privada para la administración de DAX, es posible que deba crear puntos de conexión de VPC adicionales para servicios como `com.amazonaws.region.console` y los servicios relacionados.

- Se le cobrará por la creación y el uso de cada punto de conexión de interfaz a DAX. Para obtener información sobre precios, consulte Precios de AWS PrivateLink.

Cómo funciona AWS PrivateLink con DAX

Al crear un punto de conexión de interfaz para DAX:

1. AWS crea una interfaz de red de punto de conexión en cada subred habilitada para el punto de conexión de interfaz.
2. Se trata de interfaces de red administradas por el solicitante que sirven como punto de entrada para el tráfico destinado para DAX.
3. A continuación, puede acceder a DAX a través de direcciones IP privadas dentro de la VPC.
4. Esta arquitectura le permite utilizar grupos de seguridad de VPC para administrar el acceso a los puntos de conexión.
5. Las aplicaciones pueden acceder a DynamoDB y DAX a través de sus respectivos puntos de conexión de interfaz dentro de una VPC, al tiempo que permiten que las aplicaciones en las instalaciones se conecten a través de Direct Connect o VPN.
6. Esto proporciona un modelo de conectividad uniforme en ambos servicios, simplifica la arquitectura y mejora la seguridad al mantener el tráfico dentro de la red de AWS.

Creación de puntos de conexión de interfaz para DAX

Puede crear un punto de conexión de interfaz para conectarse a DAX mediante la Consola de administración de AWS, AWS SDK, CloudFormation o la API de AWS.

Creación de un punto de conexión de interfaz para DAX con la consola

1. Vaya a la consola de Amazon VPC en https://console.aws.amazon.com/vpc/.
2. En el panel de navegación, elija Puntos de conexión.
3. Elija Crear punto de conexión.
4. Para la categoría de servicio, elija Servicios de AWS y, para el nombre del servicio, busque y seleccione `com.amazonaws.region.dax`.
5. Para la VPC, seleccione la VPC desde la que quiere acceder a DAX y, para las subredes, seleccione las subredes en las que AWS creará las interfaces de red de los puntos de conexión.

6. Para Grupos de seguridad, seleccione o cree los grupos de seguridad que deban asociarse a las interfaces de red del punto de conexión.
7. En el caso de la política, mantenga el acceso total predeterminado o personalícelo según sea necesario.
8. Seleccione Habilitar nombre de DNS para habilitar un DNS privado para el punto de conexión. Mantenga el nombre de DNS privado habilitado para evitar cambios en la configuración del SDK. Cuando está habilitado, las aplicaciones pueden seguir utilizando el nombre DNS del servicio estándar (ejemplo: `dax.region.amazonaws.com`). AWS crea una zona alojada privada en la VPC que resuelve este nombre en la dirección IP privada del punto de conexión.

   > **Note**
   >
   > Utilice nombres DNS regionales si es necesario. No se recomienda el uso de nombres DNS zonales. Además, seleccione subredes de 3 o más zonas de disponibilidad para garantizar la máxima disponibilidad a través de PrivateLink.

9. Seleccione Crear punto de conexión.

Creación de un punto de conexión de interfaz para DAX con la AWS CLI

Use el comando `create-vpc-endpoint` con el parámetro `vpc-endpoint-type` establecido en `Interface` y el parámetro `service-name` establecido en `com.amazonaws.region.dax`.

```
aws ec2 create-vpc-endpoint \
    --vpc-id vpc-ec43eb89 \
    --vpc-endpoint-type Interface \
    --service-name com.amazonaws.us-east-1.dax \
    --subnet-ids subnet-abcd1234 subnet-1a2b3c4d \
    --security-group-ids sg-1a2b3c4d \
    --private-dns-enabled
```

Recursos adicionales

Para obtener más información sobre AWS PrivateLink y puntos de conexión de VPC, consulte los siguientes recursos:

- AWS PrivateLink para DynamoDB
- AWS PrivateLink para DynamoDB Streams

- Conectar la VPC a los servicios mediante AWS PrivateLink
- Simplificación de la conectividad privada a DynamoDB con AWS PrivateLink
- AWS PrivateLink Documento técnico de

Configuración y análisis de vulnerabilidades en Amazon DynamoDB

AWS gestiona las tareas de seguridad básicas, como la aplicación de parches en la base de datos y el sistema operativo (SO) de invitado, la configuración del firewall y la recuperación de desastres. Estos procedimientos han sido revisados y certificados por los terceros pertinentes. Para obtener más detalles, consulte los siguientes recursos de :

- Validación de la conformidad para Amazon DynamoDB
- Modelo de responsabilidad compartida
- Amazon Web Services: información general de procesos de seguridad (documento técnico)

Las siguientes prácticas recomendadas sobre seguridad también evalúan la configuración y los análisis de vulnerabilidades en Amazon DynamoDB:

- Monitoree la conformidad de DynamoDB con Reglas de AWS Config
- monitorea la configuración de DynamoDB con AWS Config

Prácticas recomendadas de seguridad para Amazon DynamoDB

Amazon DynamoDB proporciona una serie de características de seguridad que debe tener en cuenta a la hora de desarrollar e implementar sus propias políticas de seguridad. Las siguientes prácticas recomendadas son directrices generales y no constituyen una solución de seguridad completa. Puesto que es posible que estas prácticas recomendadas no sean adecuadas o suficientes para el entorno, considérelas como consideraciones útiles en lugar de como normas.

Temas

- Prácticas recomendadas de seguridad preventivas de DynamoDB
- Prácticas recomendadas de detección de seguridad en DynamoDB

Prácticas recomendadas de seguridad preventivas de DynamoDB

Las siguientes prácticas recomendadas pueden serle de utilidad para evitar incidentes de seguridad en Amazon DynamoDB.

Cifrado en reposo

DynamoDB cifra en reposo todos los datos de usuario almacenados en tablas, índices, flujos y copias de seguridad mediante claves de cifrado almacenadas en AWS Key Management Service (AWS KMS). Esto proporciona una capa adicional de protección de datos al proteger los datos del acceso no autorizado al almacenamiento subyacente.

Puede especificar si DynamoDB debe utilizar una Clave propiedad de AWS (tipo de cifrado predeterminado), una Clave administrada de AWS, o una clave administrada por el cliente para cifrar los datos del usuario. Para obtener más información, consulte Cifrado en reposo de Amazon DynamoDB.

Usar roles de IAM para autenticar el acceso a DynamoDB

Para acceder a DynamoDB, los usuarios, las aplicaciones y otros servicios de AWStienen que incluir credenciales válidas de AWS en sus solicitudes API de AWS. No debe almacenar las credenciales de AWS de forma directa en la aplicación ni en una instancia EC2. Estas son las credenciales a largo plazo que no rotan automáticamente, por lo tanto, podrían tener un impacto empresarial significativo si se comprometen. Un rol de IAM lo habilita a obtener claves de acceso temporal que se pueden utilizar para tener acceso a los servicios y recursos de AWS.

Para obtener más información, consulte Identity and Access Management en Amazon DynamoDB.

Usar directivas de IAM para la autorización base de DynamoDB

Cuando concede permisos, usted decide quién debe obtenerlos, para qué API de DynamoDB se obtienen y qué acciones específicas desea permitir en esos recursos. La implementación de privilegios mínimos es la clave a la hora de reducir los riesgos de seguridad y el impacto que podrían causar los errores o los intentos malintencionados.

Asociar políticas de permisos a identidades de IAM (es decir, usuarios, grupos y roles) y, de ese modo, conceder permisos para realizar operaciones en recursos de DynamoDB.

Para hacerlo, utilice lo siguiente:

- AWS Políticas administradas por (predefinidas

- Políticas administradas por el cliente

Uso de condiciones de políticas de IAM para control de acceso preciso

Al conceder permisos en DynamoDB, puede especificar las condiciones que determinan cómo se aplica una política de permisos. La implementación de privilegios mínimos es la clave a la hora de reducir los riesgos de seguridad y el impacto que podrían causar los errores o los intentos malintencionados.

Puede especificar las condiciones al conceder permisos utilizando la política de IAM. Por ejemplo, puede hacer lo siguiente:

- Conceder permisos para que los usuarios puedan obtener acceso de solo lectura a determinados elementos y atributos de una tabla o un índice secundario.
- Conceder permisos para que los usuarios puedan obtener acceso de solo escritura a determinados atributos de una tabla, según la identidad del usuario en cuestión.

Para obtener más información, consulte Uso de condiciones de políticas de IAM para control de acceso preciso.

Utilizar un punto de enlace de la VPC y políticas para acceder a DynamoDB

Si solo necesita acceso a DynamoDB desde una Virtual Private Cloud (VPC), debe usar un punto de enlace de la VPC para limitar el acceso solo desde la VPC requerida. Al hacer esto, impide que el tráfico atraviese la red de Internet de acceso público y esté sujeto a ese entorno.

El uso de un punto de enlace de la VPC para DynamoDB le permite controlar y limitar el acceso mediante lo siguiente:

- Políticas del punto de enlace de la VPC: estas políticas se aplican en el punto de enlace de la VPC de DynamoDB. Le permiten controlar y limitar el acceso de la API a la tabla de DynamoDB.
- Políticas de IAM: mediante el uso de la condición `aws:sourceVpce` en las políticas asociadas a los usuarios, grupos o roles, puede exigir que todo el acceso a la tabla de DynamoDB se realice a través del punto de conexión de VPC especificado.

Para obtener más información, consulte Puntos de enlace para Amazon DynamoDB.

Tenga en cuenta el cifrado del lado del cliente

Le recomendamos que planifique la estrategia de cifrado antes de implementar la tabla en DynamoDB. Si almacena datos confidenciales en DynamoDB, considere incluir el cifrado del cliente en el plan. De esta forma, puede cifrar los datos lo más cerca posible del origen y

garantizar la protección durante todo el ciclo de vida. El cifrado de su información confidencial en tránsito y en reposo ayuda a garantizar que los datos de texto no cifrado no estén disponibles para ningún tercero.

El SDK de cifrado de base de datos de AWS para DynamoDB es una biblioteca de software que le ayuda a proteger los datos de la tabla antes de enviarlos a DynamoDB. Cifra, firma, verifica y descifra los elementos de la tabla de DynamoDB. Usted controla qué atributos se cifran y se firman.

Consideraciones sobre la clave principal

No utilice nombres confidenciales ni datos confidenciales en texto sin formato en la clave principal para la tabla y los índices secundarios globales. Los nombres de la clave aparecerán en la definición de la tabla. Por ejemplo, cualquier persona que tenga permisos para llamar a DescribeTable puede acceder a los nombres de la clave principal. Los valores de la clave pueden aparecer en AWS CloudTrail y en otros registros. Además, DynamoDB utiliza los valores de la clave para distribuir los datos y enrutar las solicitudes, y los administradores de AWS pueden observar los valores para mantener el buen estado del servicio.

Si necesita utilizar datos confidenciales en los valores de la tabla o las claves de GSI, le recomendamos que utilice el cifrado de cliente de extremo a extremo. Esto le permite crear referencias de clave-valor a sus datos y, al mismo tiempo, garantizar que nunca aparezcan sin cifrar en los registros relacionados con DynamoDB. Una forma de hacer esto consiste en utilizar el SDK de cifrado de bases de datos de AWS para DynamoDB, pero no es obligatorio. Si utiliza su propia solución, siempre debe utilizar un algoritmo de cifrado que sea lo suficientemente seguro. No debe utilizar una opción no criptográfica como un hash, ya que no se considera lo suficientemente segura en la mayoría de las situaciones.

Si los nombres de clave principal son confidenciales, le recomendamos que utilice `pk` y `sk` en su lugar. Se trata de una práctica recomendada general que flexibiliza el diseño de la clave de partición.

Consulte siempre a sus expertos en seguridad o al equipo de cuentas de AWS si no sabe cuál sería la opción correcta.

Prácticas recomendadas de detección de seguridad en DynamoDB

Las siguientes prácticas recomendadas para Amazon DynamoDB le pueden ser de utilidad para detectar los incidentes y los posibles puntos débiles de la seguridad.

Utilice AWS CloudTrail para monitorear el uso de la clave KMS administrada por AWS

Si está utilizando una Clave administrada de AWS para el cifrado en reposo, el uso de esta clave queda registrado en AWS CloudTrail. CloudTrail proporciona visibilidad de la actividad del usuario mediante el registro de las acciones realizadas en su cuenta. CloudTrail registra información importante acerca de cada acción, incluyendo quién hizo la solicitud, los servicios utilizados, las acciones realizadas, los parámetros de las acciones y los elementos de respuesta devueltos por el servicio de AWS. Esta información le ayuda a realizar un seguimiento de los cambios realizados en sus recursos de AWS y solucionar problemas operativos. CloudTrail facilita la conformidad con las políticas internas y las normas reglamentarias.

Puede utilizar CloudTrail para auditar el uso de las claves. CloudTrail crea archivos de registro que contienen un historial de las llamadas a la API de AWS y de los eventos relacionados de su cuenta. Estos archivos de registro incluyen todas las solicitudes a la API de AWS KMS realizadas con la Consola de administración de AWS, los SDK de AWS y las herramientas de la línea de comandos, así como las efectuadas a través de los servicios de AWS integrados. Puede utilizar estos archivos de registro para obtener información sobre cuándo se utilizó la clave KMS, la operación que se solicitó, la identidad del solicitante, la dirección IP de la que procede la solicitud, etc. Para obtener más información, consulte Registro de llamadas a la API de AWS KMS con AWS CloudTrail en la Guía del usuario de AWS CloudTrail.

Monitorear las operaciones de DynamoDB con CloudTrail

CloudTrail puede monitorear tanto los eventos del plano de control como los eventos del plano de datos. Las operaciones del plano de control le permiten crear y administrar tablas de DynamoDB. También permiten usar índices, secuencias y otros objetos que dependen de las tablas. Las operaciones del plano de datos le permiten llevar a cabo acciones de creación, lectura, actualización y eliminación (también denominadas CRUD, por sus siglas en inglés) con los datos de una tabla. Algunas de las operaciones del plano de datos permiten también leer datos de un índice secundario. Para habilitar el registro de eventos de plano de datos en CloudTrail, deberá habilitar el registro de la actividad de la API del plano de datos en CloudTrail. Para obtener más información, consulte Registro de eventos de datos para seguimiento.

Cuando se produce una actividad en DynamoDB, esa actividad se registra en un evento de CloudTrail junto con otros eventos de servicio de AWS en el historial de eventos. Para obtener más información, consulte Registro de operaciones de DynamoDB mediante AWS CloudTrail. Puede ver, buscar y descargar los últimos eventos de la cuenta de AWS. Para obtener más información, consulte Ver eventos con el historial de eventos de CloudTrail en la Guía del usuario de AWS CloudTrail.

Para mantener un registro continuo de eventos en la cuenta de AWS, incluyendo los eventos de DynamoDB, cree un seguimiento. Un seguimiento habilita a CloudTrail a enviar archivos de registro a un bucket de Amazon Simple Storage Service (Amazon S3). De forma predeterminada, cuando se crea un registro de seguimiento en la consola, este se aplica a todas las regiones de AWS. El registro de seguimiento registra los eventos de todas las regiones de la partición de AWS y envía los archivos de registro al bucket de S3 especificado. También es posible configurar otros servicios de AWS para analizar en profundidad y actuar en función de los datos de eventos recopilados en los registros de CloudTrail.

Utilice DynamoDB Streams para monitorear las operaciones del plano de datos

DynamoDB se integra con AWS Lambda para que pueda crear desencadenadores, a saber, fragmentos de código que responden automáticamente a los eventos de DynamoDB Streams. Con los desencadenadores, puede crear aplicaciones que reaccionan ante las modificaciones de datos en las tablas de DynamoDB.

Si habilita DynamoDB Streams en una tabla, puede asociar el nombre de recurso de Amazon (ARN) de la secuencia con una función de Lambda que haya escrito. Inmediatamente después de modificar un elemento de la tabla, aparece un nuevo registro en la secuencia de la tabla. AWS Lambda sondea la secuencia y llama a la función Lambda de forma sincrónica cuando detecta nuevos registros en él. La función Lambda pueden realizar cualquier acción que usted especifique, como, por ejemplo, enviar una notificación o iniciar un flujo de trabajo.

Para ver un ejemplo consulte Tutorial: uso de AWS Lambda con Amazon DynamoDB Streams. En este ejemplo se recibe una entrada de evento de DynamoDB, se procesan los mensajes que contiene y se escriben algunos de los datos de eventos entrantes en los Amazon CloudWatch Logs.

monitorea la configuración de DynamoDB con AWS Config

Utilizando AWS Config, puede monitorear y registrar continuamente los cambios de configuración de recurso de AWS. También puede utilizar AWS Config para realizar un inventario sur recursos de AWS. Cuando se detecta un cambio con respecto a un estado anterior, se puede enviar una notificación de Amazon Simple Notification Service (Amazon SNS) para que la revise y tome medidas. Siga la guía de Configuración de AWS Config con la consola, asegurando que se incluyan los tipos de recursos de DynamoDB.

Puede configurar AWS Config para que transmita cambios de configuración y notificaciones a un tema de Amazon SNS. Por ejemplo, cuando se actualiza un recurso, puede obtener una notificación enviada a su correo electrónico, para que pueda ver los cambios. También se le

puede notificar cuando AWS Config evalúa las reglas personalizadas o administradas con respecto a sus recursos.

Para ver un ejemplo, consulte Notificaciones que AWS Config envía a un tema de Amazon SNS en la Guía para desarrolladores AWS Config.

monitorea el cumplimiento de DynamoDB de las reglas AWS Config

AWS Config realiza un seguimiento constante de los cambios de configuración que se producen entre los recursos. Comprueba si estos cambios infringen cualquiera de las condiciones de sus reglas. Si un recurso infringe una regla, AWS Config marca el recurso y la regla como no conformes.

Al utilizar AWS Config para evaluar las configuraciones de los recursos, puede evaluar en qué medida las configuraciones de los recursos cumplen las prácticas internas, las directrices del sector y las normativas. AWS Config proporciona Reglas administradas de AWS, que son reglas predefinidas y personalizables que AWS Config utiliza para evaluar si los recursos de AWS cumplen con las prácticas recomendadas comunes.

Etiquetar sus recursos de DynamoDB para su identificación y automatización

Puede asignar metadatos a los recursos de AWS en forma de etiquetas. Cada etiqueta es una marca que consta de una clave definida por el cliente y un valor opcional que puede hacer que sea más fácil administrar, buscar y filtrar recursos.

El etiquetado permite implementar controles agrupados. Aunque no hay tipos inherentes de etiquetas, lo habilitan a clasificar los recursos según su finalidad, propietario, entorno u otros criterios. A continuación se muestran algunos ejemplos:

- Seguridad: se utiliza para determinar requisitos tales como el cifrado.
- Confidencialidad: un identificador para el nivel concreto de confidencialidad de los datos que admite un recurso.
- Entorno: utilizado para distinguir entre el desarrollo, la prueba y la infraestructura de producción.

Para obtener más información, consulte Estrategias de etiquetado de AWS y Etiquetado de DynamoDB.

Monitoree el uso de Amazon DynamoDB en relación con las mejores prácticas de seguridad con AWS Security Hub CSPM.

Security Hub utiliza controles de seguridad para evaluar las configuraciones de los recursos y los estándares de seguridad para ayudarle a cumplir varios marcos de conformidad.

Para obtener más información sobre el uso de Security Hub para evaluar los recursos de DynamoDB, consulte Controles de Amazon DynamoDB en la Guía del usuario de AWS Security Hub CSPM.

Monitoreo y registro en DynamoDB

El monitoreo es una parte importante del mantenimiento de la fiabilidad, la disponibilidad y el rendimiento de DynamoDB y sus soluciones de AWS. Debería recopilar datos de monitoreo de todas las partes de sus soluciones de AWS para poder depurar fácilmente un fallo multipunto.

Temas

- Plan de monitoreo
- Referencia de rendimiento
- Servicios integrados
- Herramientas de supervisión automatizadas
- Supervisión de métricas en DynamoDB con Amazon CloudWatch
- Registrar las operaciones de DynamoDB mediante AWS CloudTrail
- Análisis del acceso a los datos mediante CloudWatch contributor insights for DynamoDB

Plan de monitoreo

Antes de comenzar a monitorear DynamoDB, cree un plan de monitoreo que incluya respuestas a las siguientes preguntas:

- ¿Cuáles son los objetivos de la supervisión?
- ¿Qué recursos va a supervisar?
- ¿Con qué frecuencia va a supervisar estos recursos?
- ¿Qué herramientas de supervisión va a utilizar?
- ¿Quién se encargará de realizar las tareas de supervisión?
- ¿Quién debería recibir una notificación cuando surjan problemas?

Referencia de rendimiento

Establezca un punto de referencia del rendimiento de DynamoDB normal en su entorno. Para ello, mida el rendimiento en distintos momentos y con distintas condiciones de carga. Cuando monitoree DynamoDB , debe tener en cuenta el almacenamiento de los datos históricos de monitoreo. Estos datos almacenados le darán un punto de referencia con el que comparar los datos de desempeño

actuales, identificar los patrones de desempeño normales y las anomalías de desempeño, así como desarrollar métodos de resolución de problemas. Para establecer un punto de referencia debe, como mínimo, monitorizar los elementos siguientes:

- El número de unidades de capacidad de lectura o escritura usadas durante el periodo de tiempo especificado, para que pueda saber cuánta capacidad de desempeño provisionada se usa.
- Las solicitudes que superan la capacidad aprovisionada de lectura o escritura de una tabla durante el periodo especificado, para que pueda determinar qué solicitudes superan las cuotas de rendimiento aprovisionado de una tabla.
- Los errores del sistema, para poder determinar si alguna solicitud ha dado lugar a un error.

Servicios integrados

DynamoDB monitorea automáticamente las tablas en su nombre y notifica las métricas a través de Amazon CloudWatch. Además, DynamoDB se integra con los siguientes Servicios de AWS para ayudarlo a monitorear y solucionar problemas de sus recursos de DynamoDB.

- AWS CloudTrail captura las llamadas a la API y otros eventos relacionados que realiza la Cuenta de AWS o que se realizan en nombre de esta. Además, entrega los archivos de registro a un bucket de Amazon S3 especificado. Para obtener más información, consulte Registrar las operaciones de DynamoDB mediante AWS CloudTrail.
- Información de colaboradores es una herramienta de diagnóstico que permite identificar de un vistazo las claves de acceso más frecuentes y sometidas a más limitaciones de una tabla o índice. Para obtener más información, consulte Análisis del acceso a los datos mediante CloudWatch contributor insights for DynamoDB.

Herramientas de supervisión automatizadas

AWS proporciona diversas herramientas que puede utilizar para monitorear DynamoDB. Le recomendamos que automatice las tareas de supervisión en la medida de lo posible. Puede utilizar las siguientes herramientas de monitoreo automatizadas para vigilar a DynamoDB e informar cuando haya algún problema:

- Alarmas de Amazon CloudWatch: vigile una sola métrica durante el período de tiempo que especifique y realice una o varias acciones según el valor que muestre la métrica en relación con un determinado umbral durante una serie de períodos de tiempo.

La acción es una notificación enviada a un tema de Amazon Simple Notification Service (Amazon SNS) o a una política de Amazon EC2 Auto Scaling. Las alarmas de Amazon CloudWatch no invocan acciones tan solo por tener un estado determinado; es necesario que el estado haya cambiado y se mantenga durante un número específico de periodos. Para obtener más información, consulte Supervisión de métricas en DynamoDB con Amazon CloudWatch.

- Monitoreo de registros de AWS CloudTrail: comparta archivos de registro entre cuentas, monitoree archivos de registro de AWS CloudTrail en tiempo real enviándolas a registros de AWS CloudTrail, escriba aplicaciones de procesamiento de registros en Java y valide que sus archivos de registro no hayan cambiado después de que AWS CloudTrail los entregue. Para obtener más información, consulte ¿Qué es Registros de Amazon CloudWatch? en la Guía de usuario de AWS CloudTrail.

Supervisión de métricas en DynamoDB con Amazon CloudWatch

Puede supervisar DynamoDB con CloudWatch, que recopila y procesa los datos sin procesar de DynamoDB y los convierte en métricas legibles casi en tiempo real. Estas estadísticas se retienen durante un tiempo, para que pueda acceder a la información histórica y obtener una mejor perspectiva sobre el rendimiento de su servicio o aplicación web. De forma predeterminada, los datos de las métricas de DynamoDB se envían a CloudWatch automáticamente. Para obtener más información, consulte ¿Qué es Amazon CloudWatch? y Retención de métricas en la Guía del usuario de Amazon CloudWatch.

Temas

- ¿Cómo uso las métricas de DynamoDB?
- Visualizar métricas en la consola de CloudWatch
- Visualización de las métricas en la AWS CLI
- Dimensiones y métricas de DynamoDB
- Creación de alarmas de CloudWatch en DynamoDB

¿Cómo uso las métricas de DynamoDB?

Las métricas mostradas por DynamoDB proporcionan información que puede analizar de diferentes maneras. En la siguiente lista se indican algunos usos frecuentes de las métricas. Se trata de sugerencias que puede usar como punto de partida y no de una lista completa.

¿Cómo uso las métricas de DynamoDB?

¿Cómo...?	Métricas relevantes
¿Cómo puedo monitorear la tasa de eliminaciones de TTL en la tabla?	Puede monitorear `TimeToLiveDeletedItemCount` durante el periodo de tiempo especificado para realizar el seguimien to de la tasa de eliminaciones de TTL en la tabla. Para ver un ejemplo de una aplicación sin servidor que usa la métrica `TimeToLiveDeletedItemCount` , consulte Automatically archive items to S3 using DynamoDB time to live (TTL) with AWS Lambda and Amazon Data Firehose.
¿Cómo puedo determinar qué parte del rendimiento aprovisio nado se está utilizando?	Puede monitorizar `ConsumedReadCapacityUnits` o `ConsumedWriteCapacityUnits` durante el periodo de tiempo especificado para realizar el seguimiento de la cantidad de desempeño provisionado que se está usando.
¿Cómo puedo determinar qué solicitudes superan las cuotas de rendimiento aprovisionado de una tabla?	`ThrottledRequests` se incrementa en uno si algún evento de la solicitud supera la cuota de rendimiento aprovisionada. A continuación, para saber a qué evento de la solicitud se le está imponiendo la limitación controlada, compare las métricas `ThrottledRequests` con `ReadThrottleEvents` y `WriteThrottleEvents` para la tabla y sus índices.
¿Cómo puedo determinar si se ha producido algún error en el sistema?	Puede monitorear `SystemErrors` para determinar si alguna solicitud ha dado lugar a un código HTTP 500 (error de servidor) . Normalmente, esta métrica debe ser igual a cero. si no lo es, es conveniente investigar por qué.
¿Cómo puedo monitorear el valor de latencia de las operaciones de mi tabla?	Puede supervisar `SuccessfulRequestLatency` mediante el seguimiento de la latencia media y la latencia mediana a través de métricas de percentil (p50). Los picos ocasionales de la latencia no son motivo de preocupación. No obstante, si la latencia media o p50 (mediana) es alta, es posible que haya un problema subyacente que se debe solucionar. Para obtener más información, consulte Solución de problemas de latencia en Amazon DynamoDB.

Visualizar métricas en la consola de CloudWatch

Las métricas se agrupan primero por el espacio de nombres de servicio y, a continuación, por las diversas combinaciones de dimensiones dentro de cada espacio de nombres.

Para ver las métricas en la consola de CloudWatch

1. Abra la consola de CloudWatch en https://console.aws.amazon.com/cloudwatch/.
2. En el panel de navegación, seleccione Métricas, Todas las métricas.
3. Seleccione el espacio de nombres DynamoDB. También puede seleccionar el espacio de nombres Uso para ver las métricas de uso de DynamoDB. Para obtener más información sobre las métricas de uso, consulte Métricas de uso de AWS.
4. En la pestaña Explorar aparecen todas las métricas del espacio de nombres.
5. (Opcional) Para agregar este gráfico a un panel de CloudWatch, elija Acciones, Agregar al panel.

Visualización de las métricas en la AWS CLI

Para obtener información sobre métricas con la AWS CLI, utilice el comando de CloudWatch `list-metrics`. En el siguiente ejemplo, se enumeran todas las métricas del espacio de nombres de `AWS/DynamoDB`.

```
aws cloudwatch list-metrics --namespace "AWS/DynamoDB"
```

Para obtener estadísticas de métricas, utilice el comando `get-metric-statistics`. El siguiente comando obtiene estadísticas de `ConsumedReadCapacityUnits` para la tabla `ProductCatalog` en el periodo específico de 24 horas, con un grado de detalle de 5 minutos.

```
aws cloudwatch get-metric-statistics –namespace AWS/DynamoDB \
     –metric-name ConsumedReadCapacityUnits \
     –start-time 2023-11-01T00:00:00Z \
     –end-time 2023-11-02T00:00:00Z \
     –period 360 \
     –statistics Average \
     –dimensions Name=TableName,Value=ProductCatalog
```

El resultado de ejemplo aparece como se muestra a continuación:

```
{
    "Datapoints": [
        {
            "Timestamp": "2023-11-01T 09:18:00+00:00",
            "Average": 20,
            "Unit": "Count"
        },
        {
            "Timestamp": "2023-11-01T 04:36:00+00:00",
            "Average": 22.5,
            "Unit": "Count"
        },
        {
            "Timestamp": "2023-11-01T 15:12:00+00:00",
            "Average": 20,
            "Unit": "Count"
        }, ...
        {
            "Timestamp": "2023-11-01T 17:30:00+00:00",
            "Average": 25,
            "Unit": "Count"
        }
    ],
    "Label": " ConsumedReadCapacityUnits "
}
```

Dimensiones y métricas de DynamoDB

Cuando se interactúa con DynamoDB, este envía métricas y dimensiones a CloudWatch.

Las salidas de DynamoDB consumieron el rendimiento aprovisionado durante periodos de un minuto. El escalado automático se activa cuando la capacidad consumida supera el objetivo de utilización configurado durante dos minutos consecutivos. Es posible que las alarmas de CloudWatch tarden unos minutos antes de desencadenar el escalado automático. Este retraso garantiza una evaluación precisa de las métricas de CloudWatch. Si los picos de rendimiento consumidos están separados por más de un minuto, es posible que no se desencadene el escalado automático. Del mismo modo, se puede producir un evento de reducción vertical cuando 15 puntos de datos consecutivos sean inferiores a la utilización objetivo. En cualquier caso, tras desencadenar el escalado automático, se invoca la API UpdateTable. La actualización de la capacidad aprovisionada para la tabla o el

índice puede tardar unos minutos. Durante este periodo, cualquier solicitud que supere la capacidad aprovisionada previamente de las tablas se limita.

Visualización de métricas y dimensiones

CloudWatch muestra las siguientes métricas para DynamoDB:

Métricas de DynamoDB

> **Note**
>
> Amazon CloudWatch acumula estas métricas a intervalos de un minuto:
>
> - `ConditionalCheckFailedRequests`
> - `ConsumedReadCapacityUnits`
> - `ConsumedWriteCapacityUnits`
> - `ReadAccountLimitThrottleEvents`
> - `ReadKeyRangeThroughputThrottleEvents`
> - `ReadMaxOnDemandThroughputThrottleEvents`
> - `ReadProvisionedThroughputThrottleEvents`
> - `ReadThrottleEvents`
> - `ReturnedBytes`
> - `ReturnedItemCount`
> - `ReturnedRecordsCount`
> - `SuccessfulRequestLatency`
> - `SystemErrors`
> - `TimeToLiveDeletedItemCount`
> - `ThrottledRequests`
> - `TransactionConflict`
> - `UserErrors`
> - `WriteAccountLimitThrottleEvents`
> - `WriteKeyRangeThroughputThrottleEvents`
> - `WriteMaxOnDemandThroughputThrottleEvents`
> - `WriteProvisionedThroughputThrottleEvents`

- `WriteThrottleEvents`

Para todas las otras métricas de DynamoDB, la granularidad de acumulación es de cinco minutos.

No todas las estadísticas, tales como Average (Media) o Sum (Suma), son aplicables a todas las métricas. Sin embargo, todos estos valores están disponibles a través de la consola de Amazon DynamoDB o mediante la consola de CloudWatch, la AWS CLI o los SDK de AWS para todas las métricas.

En la siguiente lista, cada métrica tiene un conjunto de estadísticas válidas que son aplicables a esa métrica.

Lista de métricas disponibles

- AccountMaxReads
- AccountMaxTableLevelReads
- AccountMaxTableLevelWrites
- AccountMaxWrites
- AccountProvisionedReadCapacityUtilization
- AccountProvisionedWriteCapacityUtilization
- AgeOfOldestUnreplicatedRecord
- CondicionalCheckFailedRequests
- ConsumedChangeDataCaptureUnits
- ConsumedReadCapacityUnits
- ConsumedWriteCapacityUnits
- FailedToReplicateRecordCount
- MaxProvisionedTableReadCapacityUtilization
- MaxProvisionedTableWriteCapacityUtilization
- OnDemandMaxReadRequestUnits
- OnDemandMaxWriteRequestUnits
- OnlineIndexConsumedWriteCapacity
- OnlineIndexPercentageProgress

- OnlineIndexThrottleEvents
- PendingReplicationCount
- ProvisionedReadCapacityUnits
- ProvisionedWriteCapacityUnits
- ReadAccountLimitThrottleEvents
- ReadKeyRangeThroughputThrottleEvents
- ReadMaxOnDemandThroughputThrottleEvents
- ReadProvisionedThroughputThrottleEvents
- ReadThrottleEvents
- ReplicationLatency
- ReturnedBytes
- ReturnedItemCount
- ReturnedRecordsCount
- SuccessfulRequestLatency
- SystemErrors
- TimeToLiveDeletedItemCount
- ThrottledPutRecordCount
- ThrottledRequests
- TransactionConflict
- UserErrors
- WriteAccountLimitThrottleEvents
- WriteKeyRangeThroughputThrottleEvents
- WriteMaxOnDemandThroughputThrottleEvents
- WriteProvisionedThroughputThrottleEvents
- WriteThrottleEvents

AccountMaxReads

Número máximo de unidades de capacidad de lectura que puede utilizar una cuenta. Este límite no se aplica a las tablas bajo demanda ni a los índices secundarios globales.

Unidades: `Count`

Estadísticas válidas:

- `Maximum`: la cantidad máxima de unidades de capacidad de lectura que puede utilizar una cuenta.

AccountMaxTableLevelReads

La cantidad máxima de unidades de capacidad de lectura que puede utilizar una tabla o un índice secundario global de una cuenta. Para tablas bajo demanda, este límite es el tope máximo de unidades de solicitud de lectura que puede utilizar una tabla o un índice secundario global.

Unidades: `Count`

Estadísticas válidas:

- `Maximum`: la cantidad máxima de unidades de capacidad de lectura que puede utilizar una tabla o un índice secundario global de la cuenta.

AccountMaxTableLevelWrites

La cantidad máxima de unidades de capacidad de escritura que puede utilizar una tabla o un índice secundario global de una cuenta. Para tablas bajo demanda, este límite es el tope máximo de unidades de solicitud de escritura que puede utilizar una tabla o un índice secundario global.

Unidades: `Count`

Estadísticas válidas:

- `Maximum`: la cantidad máxima de unidades de capacidad de escritura que puede utilizar una tabla o un índice secundario global de la cuenta.

AccountMaxWrites

La cantidad máxima de unidades de capacidad de escritura que puede utilizar una cuenta. Este límite no se aplica a las tablas bajo demanda ni a los índices secundarios globales.

Unidades: `Count`

Estadísticas válidas:

- `Maximum`: la cantidad máxima de unidades de capacidad de escritura que puede utilizar una cuenta.

AccountProvisionedReadCapacityUtilization

El porcentaje de unidades de capacidad de lectura aprovisionadas utilizadas por su cuenta.

Unidades: `Percent`

Estadísticas válidas:

- `Maximum`: el porcentaje máximo de unidades de capacidad de lectura aprovisionadas utilizadas por la cuenta.
- `Minimum`: el porcentaje mínimo de unidades de capacidad de lectura aprovisionadas utilizadas por la cuenta.
- `Average`: el porcentaje promedio de unidades de capacidad de lectura aprovisionadas utilizadas por la cuenta. La métrica se publica a intervalos de cinco minutos. Por lo tanto, si ajusta rápidamente las unidades de capacidad de lectura aprovisionadas, es posible que esta estadística no refleje el promedio real.

AccountProvisionedWriteCapacityUtilization

El porcentaje de unidades de capacidad de escritura aprovisionadas utilizadas por su cuenta.

Unidades: `Percent`

Estadísticas válidas:

- `Maximum`: el porcentaje máximo de unidades de capacidad de escritura aprovisionadas utilizadas por la cuenta.
- `Minimum`: el porcentaje mínimo de unidades de capacidad de escritura aprovisionadas utilizadas por la cuenta.
- `Average`: el porcentaje promedio de unidades de capacidad de escritura aprovisionadas utilizadas por la cuenta. La métrica se publica a intervalos de cinco minutos. Por lo tanto, si ajusta rápidamente las unidades de capacidad de escritura aprovisionadas, es posible que esta estadística no refleje el promedio real.

AgeOfOldestUnreplicatedRecord

Ha aparecido por primera vez en la tabla DynamoDB el tiempo transcurrido desde que un registro que aún no se ha replicado en el flujo de datos de Kinesis.

Unidades: `Milliseconds`

Dimensiones: `TableName, DelegatedOperation`

Estadísticas válidas:

- `Maximum`.
- `Minimum`.
- `Average`.

CondicionalCheckFailedRequests

Cantidad de intentos fallidos para realizar escrituras condicionales. La operaciones `PutItem`, `UpdateItem`, y `DeleteItem` le permiten proporcionar una condición lógica que debe evaluarse como true (verdadera) antes de que la operación pueda continuar. Si se evalúa esta condición como false (falso), `ConditionalCheckFailedRequests` se incrementa en uno. `ConditionalCheckFailedRequests` también se incrementa en uno para las instrucciones PartiQL Update (actualizar) y Delete (eliminar) donde se proporciona una condición lógica y esa condición se evalúa como false (falso).

> **Note**
>
> Una escritura condicional fallida dará lugar a un error HTTP 400 (solicitud errónea). Estos eventos se reflejan en la métrica `ConditionalCheckFailedRequests`, pero no en la métrica `UserErrors`.

Unidades: `Count`

Dimensiones: `TableName`

Estadísticas válidas:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ConsumedChangeDataCaptureUnits

La cantidad de unidades de captura de datos de cambio consumidas.

Unidades: `Count`

Dimensiones: `TableName, DelegatedOperation`

Estadísticas válidas:

- `Minimum`
- `Maximum`
- `Average`

ConsumedReadCapacityUnits

La cantidad de unidades de capacidad de lectura usadas durante el periodo de tiempo especificado, tanto para capacidad aprovisionada como para capacidad bajo demanda, para que pueda saber cuánto rendimiento se usa. Puede recuperar la capacidad total de lectura consumida para una tabla y todos sus índices secundarios globales, o para un índice secundario global determinado. Para obtener más información, consulte Modo de capacidad de lectura/escritura.

La dimensión `TableName` devuelve la `ConsumedReadCapacityUnits` para la tabla, pero no para los índices secundarios globales. Para ver la `ConsumedReadCapacityUnits` para un índice secundario global, debe especificar tanto `TableName` como `GlobalSecondaryIndexName`.

> **Note**
>
> Esto significa que es posible que los picos cortos e intensos en el consumo de capacidad que duren solo un segundo no se reflejen con precisión en el gráfico de CloudWatch, lo que podría provocar una tasa de consumo aparente más baja durante ese minuto.
> Utilice la estadística `Sum` para calcular el rendimiento consumido. Por ejemplo, obtenga el valor `Sum` en un lapso de un minuto y divídalo por el número de segundos en un minuto (60) para calcular el promedio `ConsumedReadCapacityUnits` por segundo. Puede comparar el valor calculado con el valor de rendimiento aprovisionado que le proporciona a DynamoDB.

Unidades: `Count`

Dimensiones: `TableName, GlobalSecondaryIndexName`

Estadísticas válidas:

- `Minimum`: la cantidad mínima de unidades de capacidad de lectura consumidas por cualquier solicitud individual a la tabla o al índice.
- `Maximum`: la cantidad máxima de unidades de capacidad de lectura consumidas por cualquier solicitud individual a la tabla o al índice.
- `Average`: la capacidad de lectura promedio por solicitud consumida.

> **Note**
>
> El valor `Average` está influenciado por periodos de inactividad donde el valor de la muestra será cero.

- `Sum`: el total de unidades de capacidad de lectura consumidas. Esta es la estadística más útil para la métrica `ConsumedReadCapacityUnits`.
- `SampleCount`: representa la frecuencia con la que se emite la métrica. Incluso las tablas sin tráfico tendrán `SampleCount` emitido regularmente, pero los valores de muestra serán siempre cero.

> **Note**
>
> El valor `SampleCount` está influenciado por periodos de inactividad donde el valor de la muestra será cero.

ConsumedWriteCapacityUnits

La cantidad de unidades de capacidad de escritura usadas durante el periodo de tiempo especificado, tanto para capacidad aprovisionada como para capacidad bajo demanda, para que pueda saber cuánto rendimiento se usa. Puede recuperar la capacidad total de escritura consumida para una tabla y todos sus índices secundarios globales, o para un índice secundario global determinado. Para obtener más información, consulte Modo de capacidad de lectura/escritura.

La dimensión `TableName` devuelve la `ConsumedWriteCapacityUnits` para la tabla, pero no para los índices secundarios globales. Para ver la `ConsumedWriteCapacityUnits` para un índice secundario global, debe especificar tanto `TableName` como `GlobalSecondaryIndexName`.

> **Note**
>
> Utilice la estadística `Sum` para calcular el rendimiento consumido. Por ejemplo, obtenga el valor de `Sum` en un lapso de un minuto y divídalo por el número de segundos de un minuto (60) para calcular el promedio de `ConsumedWriteCapacityUnits` por segundo (reconociendo que en este promedio no se ponen de manifiesto los picos grandes pero breves en la actividad de escritura que se hayan producido durante ese minuto). Puede comparar el valor calculado con el valor de rendimiento aprovisionado que le proporciona a DynamoDB.

Unidades: `Count`

Dimensiones: `TableName, GlobalSecondaryIndexName`

Estadísticas válidas:

- `Minimum`: la cantidad mínima de unidades de capacidad de escritura consumidas por cualquier solicitud individual a la tabla o al índice.
- `Maximum`: la cantidad máxima de unidades de capacidad de escritura consumidas por cualquier solicitud individual a la tabla o al índice.
- `Average`: la capacidad de escritura promedio por solicitud consumida.

  > **Note**
  >
  > El valor `Average` está influenciado por periodos de inactividad donde el valor de la muestra será cero.

- `Sum`: el total de unidades de capacidad de escritura consumidas. Esta es la estadística más útil para la métrica `ConsumedWriteCapacityUnits`.
- `SampleCount`: representa la frecuencia con la que se emite la métrica. Incluso las tablas sin tráfico tendrán `SampleCount` emitido regularmente, pero los valores de muestra serán siempre cero.

> **Note**
>
> El valor `SampleCount` está influenciado por periodos de inactividad donde el valor de la muestra será cero.

FailedToReplicateRecordCount

Número de registros que DynamoDB no ha podido replicar en el flujo de datos de Kinesis.

Unidades: `Count`

Dimensiones: `TableName`, `DelegatedOperation`

Estadísticas válidas:

- `Sum`

MaxProvisionedTableReadCapacityUtilization

El porcentaje de capacidad de lectura aprovisionado usado por la tabla de lectura con mayor aprovisionamiento o el índice secundario global de una cuenta.

Unidades: `Percent`

Estadísticas válidas:

- `Maximum`: el porcentaje máximo de unidades de capacidad de lectura aprovisionadas utilizadas por la tabla de lectura con mayor aprovisionamiento o el índice secundario global de una cuenta.
- `Minimum`: el porcentaje mínimo de unidades de capacidad de lectura aprovisionadas utilizadas por la tabla de lectura con mayor aprovisionamiento o el índice secundario global de una cuenta.
- `Average`: el porcentaje promedio de unidades de capacidad de lectura aprovisionadas utilizadas por la tabla de lectura con mayor aprovisionamiento o el índice secundario global de la cuenta. La métrica se publica a intervalos de cinco minutos. Por lo tanto, si ajusta rápidamente las unidades de capacidad de lectura aprovisionadas, es posible que esta estadística no refleje el promedio real.

MaxProvisionedTableWriteCapacityUtilization

El porcentaje de capacidad de escritura aprovisionada utilizada por la tabla de escritura con mayor aprovisionamiento o el índice secundario global de una cuenta.

Unidades: `Percent`

Estadísticas válidas:

- `Maximum`: el porcentaje máximo de unidades de capacidad de escritura aprovisionadas utilizadas por la tabla de escritura con mayor aprovisionamiento o el índice secundario global de una cuenta.
- `Minimum`: el porcentaje mínimo de unidades de capacidad de escritura aprovisionadas utilizadas por la tabla de escritura con mayor aprovisionamiento o el índice secundario global de una cuenta.
- `Average`: el porcentaje promedio de unidades de capacidad de escritura aprovisionadas utilizadas por la tabla de escritura con mayor aprovisionamiento o el índice secundario global de la cuenta. La métrica se publica a intervalos de cinco minutos. Por lo tanto, si ajusta rápidamente las unidades de capacidad de escritura aprovisionadas, es posible que esta estadística no refleje el promedio real.

OnDemandMaxReadRequestUnits

La cantidad de unidades de solicitud de lectura bajo demanda especificadas para una tabla o un índice secundario global.

Para ver `OnDemandMaxReadRequestUnits` para una tabla, debe especificar `TableName`. Para ver la `OnDemandMaxReadRequestUnits` para un índice secundario global, debe especificar tanto `TableName` como `GlobalSecondaryIndexName`.

Unidades: recuento

Dimensiones: `TableName`, `GlobalSecondaryIndexName`

Estadísticas válidas:

- `Minimum`: la configuración mínima para las unidades de solicitud de lectura bajo demanda. Si utiliza `UpdateTable` para aumentar las unidades de solicitud de lectura, esta métrica muestra el valor mínimo de `ReadRequestUnits` bajo demanda durante este periodo de tiempo.
- `Maximum`: la configuración máxima para las unidades de solicitud de lectura bajo demanda. Si utiliza `UpdateTable` para reducir las unidades de solicitud de lectura, esta métrica muestra el valor máximo de `ReadRequestUnits` bajo demanda durante este periodo de tiempo.

- `Average`: el promedio de unidades de solicitud de lectura bajo demanda. La métrica `OnDemandMaxReadRequestUnits` se publica a intervalos de cinco minutos. Por lo tanto, si ajusta rápidamente las unidades de solicitud de lectura bajo demanda, es posible que esta estadística no refleje el promedio real.

OnDemandMaxWriteRequestUnits

La cantidad de unidades de solicitud de escritura bajo demanda especificadas para una tabla o un índice secundario global.

Para ver `OnDemandMaxWriteRequestUnits` para una tabla, debe especificar `TableName`. Para ver la `OnDemandMaxWriteRequestUnits` para un índice secundario global, debe especificar tanto `TableName` como `GlobalSecondaryIndexName`.

Unidades: `Count`

Dimensiones: `TableName`, `GlobalSecondaryIndexName`

Estadísticas válidas:

- `Minimum`: la configuración mínima para las unidades de solicitud de escritura bajo demanda. Si utiliza `UpdateTable` para aumentar las unidades de solicitud de escritura, esta métrica muestra el valor mínimo de `WriteRequestUnits` bajo demanda durante este periodo de tiempo.
- `Maximum`: la configuración máxima para las unidades de solicitud de escritura bajo demanda. Si utiliza `UpdateTable` para reducir las unidades de solicitud de escritura, esta métrica muestra el valor máximo de `WriteRequestUnits` bajo demanda durante este periodo de tiempo.
- `Average`: el promedio de unidades de solicitud de escritura bajo demanda. La métrica `OnDemandMaxWriteRequestUnits` se publica a intervalos de cinco minutos. Por lo tanto, si ajusta rápidamente las unidades de solicitud de escritura bajo demanda, es posible que esta estadística no refleje el promedio real.

OnlineIndexConsumedWriteCapacity

La cantidad de unidades de capacidad de escritura consumidas al agregar un nuevo índice secundario global a una tabla. Si la capacidad de escritura del índice es demasiado baja, es posible que se limite la actividad de escritura entrante durante la fase de replicación. Esto puede incrementar el tiempo que se tarda en crear el índice. Debe monitorear esta estadística mientras se

está creando el índice para determinar si la capacidad de escritura del índice no está suficientemente aprovisionada.

Puede ajustar la capacidad de escritura del índice mediante la operación `UpdateTable`, incluso durante la creación del índice.

La métrica `ConsumedWriteCapacityUnits` para el índice no incluye el rendimiento de escritura consumido durante la creación del índice.

> **Note**
>
> Es posible que esta métrica no se emita si la fase de reposición del nuevo índice secundario global se completa rápidamente (en menos de unos minutos), lo que puede ocurrir si la tabla base tiene pocos o ningún elemento que reponer en el índice.

Unidades: `Count`

Dimensiones: `TableName, GlobalSecondaryIndexName`

Estadísticas válidas:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

OnlineIndexPercentageProgress

El porcentaje de finalización cuando se agrega un nuevo índice secundario global a una tabla. DynamoDB primero debe asignar recursos para el nuevo índice y, a continuación, rellenar los atributos de la tabla en el índice. Para tablas grandes, este proceso puede llevar mucho tiempo. Debe monitorear esta estadística para ver el progreso relativo a medida que DynamoDB crea el índice.

Unidades: `Count`

Dimensiones: `TableName, GlobalSecondaryIndexName`

Estadísticas válidas:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

OnlineIndexThrottleEvents

La cantidad de eventos de limitación de escritura que se producen al agregar un nuevo índice secundario global a una tabla. Estos eventos indican que la creación del índice tardará más en completarse, ya que la actividad de escritura entrante excede el rendimiento de escritura aprovisionado del índice.

Puede ajustar la capacidad de escritura del índice mediante la operación `UpdateTable`, incluso durante la creación del índice.

La métrica `WriteThrottleEvents` para el índice no incluye ningún evento de limitación que se produzca durante la creación del índice.

Unidades: `Count`

Dimensiones: `TableName, GlobalSecondaryIndexName`

Estadísticas válidas:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

PendingReplicationCount

Métrica para Versión 2017.11.29 (heredada) de las tablas globales (solo tablas globales). La cantidad de actualizaciones de elementos que se escriben en la réplica de tabla pero que todavía no se han escrito en otra réplica de la tabla global.

Unidades: `Count`

Dimensiones: `TableName, ReceivingRegion`

Estadísticas válidas:

- `Average`
- `Sample Count`
- `Sum`

ProvisionedReadCapacityUnits

La cantidad de unidades de capacidad de lectura aprovisionadas para una tabla o un índice secundario global. La dimensión `TableName` devuelve la `ProvisionedReadCapacityUnits` para la tabla, pero no para los índices secundarios globales. Para ver la `ProvisionedReadCapacityUnits` para un índice secundario global, debe especificar tanto `TableName` como `GlobalSecondaryIndexName`.

Unidades: :`Count`

Dimensiones: `TableName`, `GlobalSecondaryIndexName`

Estadísticas válidas:

- `Minimum`: la configuración más baja para la capacidad de lectura aprovisionada. Si utiliza `UpdateTable` para aumentar la capacidad de lectura, esta métrica muestra el valor más bajo de `ReadCapacityUnits` aprovisionado durante este periodo de tiempo.
- `Maximum`: la configuración más alta para la capacidad de lectura aprovisionada. Si utiliza `UpdateTable` para reducir la capacidad de lectura, esta métrica muestra el valor más alto de `ReadCapacityUnits` aprovisionado durante este período de tiempo.
- `Average`: la capacidad de lectura aprovisionada promedio. La métrica `ProvisionedReadCapacityUnits` se publica a intervalos de cinco minutos. Por lo tanto, si ajusta rápidamente las unidades de capacidad de lectura aprovisionadas, es posible que esta estadística no refleje el promedio real.

ProvisionedWriteCapacityUnits

La cantidad de unidades de capacidad de escritura aprovisionadas para una tabla o un índice secundario global.

La dimensión `TableName` devuelve la `ProvisionedWriteCapacityUnits` para la tabla, pero no para los índices secundarios globales. Para ver la `ProvisionedWriteCapacityUnits` para un índice secundario global, debe especificar tanto `TableName` como `GlobalSecondaryIndexName`.

Unidades: `Count`

Dimensiones: `TableName`, `GlobalSecondaryIndexName`

Estadísticas válidas:

- `Minimum`: la configuración más baja para la capacidad de escritura aprovisionada. Si utiliza `UpdateTable` para aumentar la capacidad de escritura, esta métrica muestra el valor más bajo de `WriteCapacityUnits` aprovisionado durante este período de tiempo.
- `Maximum`: la configuración más alta para la capacidad de escritura aprovisionada. Si utiliza `UpdateTable` para reducir la capacidad de escritura, esta métrica muestra el valor más alto de `WriteCapacityUnits` aprovisionado durante este período de tiempo.
- `Average`: la capacidad de escritura aprovisionada promedio. La métrica `ProvisionedWriteCapacityUnits` se publica a intervalos de cinco minutos. Por lo tanto, si ajusta rápidamente las unidades de capacidad de escritura aprovisionadas, es posible que esta estadística no refleje el promedio real.

ReadAccountLimitThrottleEvents

El número de solicitudes de lectura limitadas debido a los límites de la cuenta.

Unidades: `Count`

Dimensiones: `TableName`, `GlobalSecondaryIndexName`

Estadísticas válidas:

- `Sum`: el número total de eventos limitados.
- `SampleCount`: número de casos de limitación.
- `Minimum`: número mínimo de eventos limitados en un ejemplo determinado.
- `Maximum`: número máximo de eventos limitados en un ejemplo determinado.

ReadKeyRangeThroughputThrottleEvents

El número de solicitudes de lectura limitadas debido a los límites de las particiones.

Unidades: `Count`

Dimensiones: `TableName`, `GlobalSecondaryIndexName`

Estadísticas válidas:

- `Sum`: el número total de eventos limitados.
- `SampleCount`: número de casos de limitación.
- `Minimum`: número mínimo de eventos limitados en un ejemplo determinado.
- `Maximum`: número máximo de eventos limitados en un ejemplo determinado.

ReadMaxOnDemandThroughputThrottleEvents

El número de solicitudes de lectura limitadas debido al rendimiento máximo bajo demanda.

Unidades: `Count`

Dimensiones: `TableName`, `GlobalSecondaryIndexName`

Estadísticas válidas:

- `Sum`: el número total de eventos limitados.
- `SampleCount`: número de casos de limitación.
- `Minimum`: número mínimo de eventos limitados en un ejemplo determinado.
- `Maximum`: número máximo de eventos limitados en un ejemplo determinado.

ReadProvisionedThroughputThrottleEvents

El número de solicitudes de lectura limitadas debido a los límites de rendimiento aprovisionados.

Unidades: `Count`

Dimensiones: `TableName`, `GlobalSecondaryIndexName`

Estadísticas válidas:

- `Sum`: el número total de eventos limitados.
- `SampleCount`: número de casos de limitación.
- `Minimum`: número mínimo de eventos limitados en un ejemplo determinado.

- `Maximum`: número máximo de eventos limitados en un ejemplo determinado.

ReadThrottleEvents

Solicitud a DynamoDB que exceden a las unidades de capacidad de lectura aprovisionadas para una tabla o un índice secundario global.

Una sola solicitud puede dar lugar a múltiples eventos. Por ejemplo, un `BatchGetItem` que lea 10 elementos se procesa como 10 eventos de `GetItem`. Para cada evento, `ReadThrottleEvents` se incrementa en uno si ese evento está limitado. La métrica `ThrottledRequests` para todo el `BatchGetItem` no se incrementa a menos que se limiten los 10 eventos de `GetItem`.

La dimensión `TableName` devuelve la `ReadThrottleEvents` para la tabla, pero no para los índices secundarios globales. Para ver la `ReadThrottleEvents` para un índice secundario global, debe especificar tanto `TableName` como `GlobalSecondaryIndexName`.

Unidades: `Count`

Dimensiones: `TableName`, `GlobalSecondaryIndexName`

Estadísticas válidas:

- `SampleCount`
- `Sum`

ReplicationLatency

(Esta métrica es para tablas globales de DynamoDB). El tiempo transcurrido entre la aparición de un elemento actualizado en la transmisión de DynamoDB para una réplica de tabla y la aparición de ese elemento en otra réplica de la tabla global.

Unidades: `Milliseconds`

Dimensiones: `TableName, ReceivingRegion`

Estadísticas válidas:

- `Average`
- `Minimum`

- `Maximum`

ReturnedBytes

La cantidad de bytes devueltos por operaciones `GetRecords` (Amazon DynamoDB Streams) durante el periodo de tiempo especificado.

Unidades: `Bytes`

Dimensiones: `Operation, StreamLabel, TableName`

Estadísticas válidas:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

ReturnedItemCount

La cantidad de elementos devueltos por las operaciones `Query`, `Scan` o `ExecuteStatement` (selecto) durante el periodo de tiempo especificado.

La cantidad de elementos devueltos no es necesariamente igual a la cantidad de elementos evaluados. Por ejemplo, suponga que solicitó un `Scan` en una tabla o en un índice que tenía 100 elementos, pero especificó un `FilterExpression` que redujo los resultados de modo que solo se devolvieron 15 artículos. En este caso, la respuesta de `Scan` va a contener un `ScanCount` de 100 y un `Count` de 15 artículos devueltos.

Unidades: `Count`

Dimensiones: `TableName, Operation`

Estadísticas válidas:

- `Minimum`
- `Maximum`
- `Average`

- `SampleCount`
- `Sum`

ReturnedRecordsCount

La cantidad de registros de transmisión devueltos por las operaciones `GetRecords` (Amazon DynamoDB Streams) durante el periodo de tiempo especificado.

Unidades: `Count`

Dimensiones: `Operation, StreamLabel, TableName`

Estadísticas válidas:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`
- `Sum`

SuccessfulRequestLatency

La latencia de las solicitudes correctas a DynamoDB o Amazon DynamoDB Streams durante el periodo de tiempo especificado. `SuccessfulRequestLatency` puede proporcionar dos tipos distintos de información:

- El tiempo transcurrido para las solicitudes correctas (`Minimum`, `Maximum`, `Sum`, `Average` o `Percentile`).
- El número de solicitudes realizadas correctamente (`SampleCount`).

`SuccessfulRequestLatency` refleja la actividad solo dentro de DynamoDB o Amazon DynamoDB Streams y no tiene en cuenta la latencia de la red ni la actividad del cliente.

> **Note**
>
> Para analizar valores de percentiles personalizados (como p99,9), puede introducir manualmente el percentil deseado (por ejemplo, p99,9) en el campo de estadísticas de

métricas de CloudWatch. Esto le permite evaluar las distribuciones de latencia más allá de los percentiles predeterminados que aparecen en la lista desplegable.

Unidades: `Milliseconds`

Dimensiones: `TableName, Operation, StreamLabel`

Estadísticas válidas:

- `Minimum`
- `Maximum`
- `Sum`
- `Average`
- `Percentile`
- `SampleCount`

SystemErrors

Las solicitudes a DynamoDB o Amazon DynamoDB Streams que generan un código de estado HTTP 500 durante el periodo de tiempo especificado. Un código HTTP 500 normalmente indica un error de servicio interno.

> **Note**
>
> Cuando DynamoDB devuelve un error del sistema (HTTP 500), la mayoría de los AWS SDK realizan automáticamente un número configurable de reintentos. Si el problema se resuelve durante un reintento, la aplicación continúa sin detectar el error y es posible que note un aumento de la latencia percibida por parte del cliente. Si el error persiste después de todos los reintentos, se propaga al código de la aplicación.

Unidades: `Count`

Dimensiones: `TableName, Operation`

Estadísticas válidas:

- `Sum`
- `SampleCount`

TimeToLiveDeletedItemCount

La cantidad de elementos eliminados por Time to Live (TTL, periodo de vida) durante el periodo de tiempo especificado. Esta métrica le ayuda a monitorear la tasa de eliminaciones de TTL en la tabla.

Unidades: `Count`

Dimensiones: TableName

Estadísticas válidas:

- `Sum`

ThrottledPutRecordCount

El número de registros que se han visto limitados por el flujo de datos de Kinesis debido a la insuficiente capacidad de Kinesis Data Streams.

Unidades: `Count`

Dimensiones: TableName, DelegatedOperation

Estadísticas válidas:

- `Minimum`
- `Maximum`
- `Average`
- `SampleCount`

ThrottledRequests

Solicitudes a DynamoDB que exceden los límites de rendimiento aprovisionado en un recurso (tales como una tabla o un índice).

`ThrottledRequests`Se incrementa en uno si algún evento de la solicitud supera el límite de rendimiento aprovisionado. Por ejemplo, si actualiza un elemento de una tabla con índices secundarios globales, hay varios eventos: una escritura en la tabla y una escritura en cada uno de

los índices. Si uno o más de estos eventos están limitados, entonces `ThrottledRequests` se incrementa en uno.

> **Note**
>
> En una solicitud por lotes (`BatchGetItem` o `BatchWriteItem`), `ThrottledRequests` solo se incrementa si todas las solicitudes en el lote están limitadas.
> Si se limita cualquier solicitud individual dentro del lote, se incrementa una de las siguientes métricas:
>
> - `ReadThrottleEvents`: para un evento `GetItem` limitado dentro de `BatchGetItem`.
> - `WriteThrottleEvents`: para un evento `PutItem` o `DeleteItem` limitado dentro de `BatchWriteItem`.

Para obtener información sobre qué evento está limitando controladamente la solicitud, compare `ThrottledRequests` con `ReadThrottleEvents` y `WriteThrottleEvents` para la tabla y sus índices.

> **Note**
>
> Una solicitud limitada dará como resultado un código de estado HTTP 400. Tales eventos se reflejan en la métrica `ThrottledRequests`, pero no en la métrica `UserErrors`.

Unidades: `Count`

Dimensiones: `TableName, Operation`

Estadísticas válidas:

- `Sum`
- `SampleCount`

TransactionConflict

Solicitudes a nivel de elemento rechazadas debido a conflictos transaccionales entre solicitudes simultáneas en los mismos elementos. Para obtener más información, consulte Gestión de conflictos de transacciones en DynamoDB.

Unidades: `Count`

Dimensiones: `TableName`

Estadísticas válidas:

- `Sum`: la cantidad de solicitudes a nivel de elemento rechazadas debido a conflictos de transacciones.

  > **Note**
  >
  > Si varias solicitudes a nivel de elemento dentro de una llamada a `TransactWriteItems` o `TransactGetItems` fueron rechazadas, `Sum` se incrementa en uno para cada solicitud a nivel de elemento `Put`, `Update`, `Delete` o `Get`.

- `SampleCount`: la cantidad de solicitudes rechazadas debido a conflictos de transacciones.

  > **Note**
  >
  > Si varias solicitudes al nivel de elemento dentro de una llamada a `TransactWriteItems` o `TransactGetItems` fueron rechazadas, `SampleCount` solo se incrementa en uno.

- `Min`: la cantidad mínima de solicitudes a nivel de elemento rechazadas dentro de una llamada a `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem` o `DeleteItem`.
- `Max` la cantidad máxima de solicitudes de nivel de elemento rechazadas dentro de una llamada a `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem` o `DeleteItem`.
- `Average` la cantidad promedio de solicitudes de nivel de elemento rechazadas dentro de una llamada a `TransactWriteItems`, `TransactGetItems`, `PutItem`, `UpdateItem` o `DeleteItem`.

UserErrors

Solicitudes a DynamoDB o Amazon DynamoDB Streams que generan un código de estado HTTP 400 durante el periodo de tiempo especificado. Un código HTTP 400 normalmente indica un error del lado del cliente, como una combinación de parámetros no válida, un intento de actualizar una tabla inexistente o una firma de solicitud incorrecta.

Algunos ejemplos de excepciones que registrarán métricas relacionadas con `UserErrors` serían:

- `ResourceNotFoundException`
- `ValidationException`
- `TransactionConflict`

Todos estos eventos se reflejan en la métrica `UserErrors`, a excepción de los siguientes elementos:

- ProvisionedThroughputExceededException: consulte la métrica `ThrottledRequests` en esta sección.
- ConditionalCheckFailedException: consulte la métrica `ConditionalCheckFailedRequests` en esta sección.

`UserErrors` representa la suma de errores HTTP 400 para las solicitudes de DynamoDB o Amazon DynamoDB Streams para la región de AWS y la cuenta de AWS actuales.

Unidades: `Count`

Estadísticas válidas:

- `Sum`
- `SampleCount`

WriteAccountLimitThrottleEvents

El número de solicitudes de escritura limitadas debido a los límites de la cuenta.

Unidades: `Count`

Dimensiones: `TableName`

Estadísticas válidas:

- `Sum`: el número total de eventos limitados.
- `SampleCount`: número de casos de limitación.
- `Minimum`: número mínimo de eventos limitados en un ejemplo determinado.
- `Maximum`: número máximo de eventos limitados en un ejemplo determinado.

WriteKeyRangeThroughputThrottleEvents

El número de solicitudes de escritura limitadas debido a los límites de las particiones.

Unidades: `Count`

Dimensiones: `TableName`, `GlobalSecondaryIndexName`

Estadísticas válidas:

- `Sum`: el número total de eventos limitados.
- `SampleCount`: número de casos de limitación.
- `Minimum`: número mínimo de eventos limitados en un ejemplo determinado.
- `Maximum`: número máximo de eventos limitados en un ejemplo determinado.

WriteMaxOnDemandThroughputThrottleEvents

El número de solicitudes de escritura se ha limitado debido al rendimiento máximo bajo demanda.

Unidades: `Count`

Dimensiones: `TableName`, `GlobalSecondaryIndexName`

Estadísticas válidas:

- `Sum`: el número total de eventos limitados.
- `SampleCount`: número de casos de limitación.
- `Minimum`: número mínimo de eventos limitados en un ejemplo determinado.
- `Maximum`: número máximo de eventos limitados en un ejemplo determinado.

WriteProvisionedThroughputThrottleEvents

El número de solicitudes de escritura limitadas debido a los límites de rendimiento aprovisionados.

Unidades: `Count`

Dimensiones: `TableName`, `GlobalSecondaryIndexName`

Estadísticas válidas:

- `Sum`: el número total de eventos limitados.
- `SampleCount`: número de casos de limitación.
- `Minimum`: número mínimo de eventos limitados en un ejemplo determinado.
- `Maximum`: número máximo de eventos limitados en un ejemplo determinado.

WriteThrottleEvents

Solicitudes a DynamoDB que exceden las unidades de capacidad de escritura aprovisionadas para una tabla o un índice secundario global.

Una sola solicitud puede dar lugar a múltiples eventos. Por ejemplo, una solicitud `PutItem` en una tabla con tres índices secundarios globales daría como resultado cuatro eventos: la escritura de la tabla y la escritura de cada uno de los tres índices. Para cada evento, la métrica `WriteThrottleEvents` se incrementa en uno si ese evento está limitado. Para una sola solicitud `PutItem`, si alguno de los eventos está limitado, `ThrottledRequests` también se incrementa en uno. Para `BatchWriteItem`, la métrica `ThrottledRequests` para toda la `BatchWriteItem` no se incrementa a menos que se limiten todos los eventos `PutItem` o `DeleteItem`.

La dimensión `TableName` devuelve la `WriteThrottleEvents` para la tabla, pero no para los índices secundarios globales. Para ver la `WriteThrottleEvents` para un índice secundario global, debe especificar tanto `TableName` como `GlobalSecondaryIndexName`.

Unidades: `Count`

Dimensiones: `TableName, GlobalSecondaryIndexName`

Estadísticas válidas:

- `Sum`
- `SampleCount`

Métricas de uso

Las métricas de uso de CloudWatch le permiten administrar el uso de forma proactiva mediante la visualización de métricas en la consola de CloudWatch, la creación de paneles personalizados, la detección de cambios en la actividad con la detección de anomalías de CloudWatch y la configuración de alarmas que le avisan cuando el uso se acerca a un umbral.

DynamoDB también integra estas métricas de uso con Service Quotas. Puede usar CloudWatch para administrar el uso de las cuotas de servicio de la cuenta. Para obtener más información acerca, consulte Visualización de las cuotas de servicio y configuración de alarmas.

Lista de métricas de uso disponibles

- AccountProvisionedWriteCapacityUnits
- AccountProvisionedReadCapacityUnits
- TableCount

AccountProvisionedWriteCapacityUnits

La suma de las unidades de capacidad de escritura aprovisionadas para todas las tablas e índices secundarios globales de una cuenta.

Unidades: `Count`

Estadísticas válidas:

- `Minimum`: el menor número de unidades de capacidad de escritura aprovisionadas durante un periodo de tiempo.
- `Maximum`: el mayor número de unidades de capacidad de escritura aprovisionadas durante un periodo de tiempo.
- `Average`: el número promedio de unidades de capacidad de escritura aprovisionadas durante un periodo de tiempo.

Esta métrica se publica a intervalos de cinco minutos. Por lo tanto, si ajusta rápidamente las unidades de capacidad de escritura aprovisionadas, es posible que esta estadística no refleje el promedio real.

AccountProvisionedReadCapacityUnits

La suma de las unidades de capacidad de lectura aprovisionadas para todas las tablas e índices secundarios globales de una cuenta.

Unidades: `Count`

Estadísticas válidas:

- `Minimum`: el menor número de unidades de capacidad de lectura aprovisionadas durante un periodo de tiempo.
- `Maximum`: el mayor número de unidades de capacidad de lectura aprovisionadas durante un periodo de tiempo.
- `Average`: el número promedio de unidades de capacidad de lectura aprovisionadas durante un periodo de tiempo.

Esta métrica se publica a intervalos de cinco minutos. Por lo tanto, si ajusta rápidamente las unidades de capacidad de lectura aprovisionadas, es posible que esta estadística no refleje el promedio real.

TableCount

Número de tablas de activas de una cuenta.

Unidades: `Count`

Estadísticas válidas:

- `Minimum`: el menor número de tablas durante un periodo de tiempo.
- `Maximum`: el mayor número de tablas durante un periodo de tiempo.
- `Average`: el número promedio de tablas durante un periodo de tiempo.

Descripción de las métricas y dimensiones de DynamoDB

Las métricas de DynamoDB se identifican por los valores de la cuenta, el nombre de la tabla, el nombre del índice secundario global o la operación. Puede usar la consola de CloudWatch para recuperar los datos de DynamoDB junto con cualquier dimensión de la siguiente tabla.

Lista de dimensiones disponibles

- DelegatedOperation
- GlobalSecondaryIndexName
- Operación
- OperationType
- Verbo
- ReceivingRegion

- StreamLabel
- TableName

DelegatedOperation

Esta dimensión limita los datos a las operaciones que DynamoDB realiza en su nombre. Las siguientes operaciones están capturadas:

- Cambie la captura de datos para Kinesis Data Streams

GlobalSecondaryIndexName

Esta dimensión limita los datos a un índice secundario global de una tabla. Si especifica `GlobalSecondaryIndexName`, también debe especificar `TableName`.

Operación

Esta dimensión limita los datos a una de las siguientes operaciones de DynamoDB:

- `PutItem`
- `DeleteItem`
- `UpdateItem`
- `GetItem`
- `BatchGetItem`
- `Scan`
- `Query`
- `BatchWriteItem`
- `TransactWriteItems`
- `TransactGetItems`
- `ExecuteTransaction`
- `BatchExecuteStatement`
- `ExecuteStatement`

Además, puede limitar los datos a la siguiente operación de Amazon DynamoDB Streams:

- `GetRecords`

OperationType

Esta dimensión limita los datos a uno de los siguientes tipos de operaciones:

- `Read`
- `Write`

Esta dimensión se emite para las solicitudes `ExecuteTransaction` y `BatchExecuteStatement`.

Verbo

Esta dimensión limita los datos a una de los siguientes verbos PartiQL de DynamoDB:

- Inserte: `PartiQLInsert`
- Seleccionar: `PartiQLSelect`
- Actualizar: `PartiQLUpdate`
- Eliminar: `PartiQLDelete`

Esta dimensión se emite para la operación `ExecuteStatement`.

ReceivingRegion

Esta dimensión limita los datos a una región de AWS particular. Se utiliza con métricas procedentes de réplicas de tablas dentro de una tabla global de DynamoDB.

StreamLabel

Esta dimensión limita los datos a una etiqueta de transmisión específica. Se utiliza con métricas procedentes de operaciones `GetRecords` de Amazon DynamoDB Streams.

TableName

Esta dimensión limita los datos a una tabla específica. Este valor puede ser cualquier nombre de tabla en la región actual y la cuenta AWS actual.

Creación de alarmas de CloudWatch en DynamoDB

Una alarma de CloudWatch vigila una única métrica durante el período especificado y realiza una o varias acciones determinadas en función del valor de la métrica en relación a un umbral durante un

período de tiempo. La acción es una notificación que se envía a un tema de Amazon SNS o a una política de Amazon EC2 Auto Scaling. Puede agregar alarmas a los paneles para poder monitorear y recibir alertas sobre sus recursos y aplicaciones de AWS en varias regiones. No existe ningún límite respecto al número de alarmas que se pueden crear. Las alarmas de CloudWatch no invocan acciones tan solo por tener un estado determinado; es necesario que el estado haya cambiado y se mantenga durante un número específico de periodos. Para obtener una lista de las alarmas de DynamoDB recomendadas, consulte Alarmas recomendadas.

> **Note**
>
> Debe especificar todas las dimensiones necesarias al crear la alarma CloudWatch, ya que CloudWatch no agrupará métricas para una dimensión que falta. Si crea una alarma de CloudWatch con una dimensión que falta no dará lugar a un error al crear la alarma.

Supongamos que tiene una tabla aprovisionada con cinco unidades de capacidad de lectura. Desea recibir una notificación antes de que consuma toda la capacidad de lectura aprovisionada, por lo que decide crear una alarma de CloudWatch para recibir una notificación cuando la capacidad consumida alcance el 80 % de lo que ha aprovisionado para la tabla. Puede crear alarmas en la consola de CloudWatch o utilizando la AWS CLI.

Creación de una alarma en la consola de CloudWatch

Creación de una alarma en la consola de CloudWatch

1. Inicie sesión en la Consola de administración de AWS y abra la consola de CloudWatch en https://console.aws.amazon.com/cloudwatch/.
2. En el panel de navegación, elija Alarms (Alarmas) y, luego, Create Alarm (Crear alarma).
3. Elija Crear alarma.
4. Busque la fila en la que está la tabla que desea monitorear y **`ConsumeReadCapacityUnits`** en la columna Nombre de métrica. Seleccione la casilla de verificación situada junto a esta fila y elija Seleccionar métrica.
5. En Especifique la métrica y las condiciones, en Estadística, elija Suma. Elija un Periodo de 1 minuto.
6. En Conditions (Condiciones), especifique lo siguiente:

    a. En Threshold type (Tipo de umbral), elija Static (Estático).

b. En Siempre que **`ConsumedReadCapacityUnits`** sea , elija Mayor/Igual y especifique un umbral de 240.

7. Elija Siguiente.
8. En Notificación, elija **`In alarm`** y seleccione el tema de SNS que enviará las notificaciones cuando la alarma se encuentre en el estado `ALARM`.
9. Cuando haya terminado, elija Next (Siguiente).
10. Ingrese un nombre y una descripción para la alarma y, a continuación, elija Next (Siguiente).
11. En Obtener vista previa y crear, confirme que la información y las condiciones son las que desea y, a continuación, elija Crear alarma.

Creación de una alarma en la AWS CLI

```
aws cloudwatch put-metric-alarm \
    -\-alarm-name ReadCapacityUnitsLimitAlarm \
    -\-alarm-description "Alarm when read capacity reaches 80% of my provisioned read
 capacity" \
    -\-namespace AWS/DynamoDB \
    -\-metric-name ConsumedReadCapacityUnits \
    -\-dimensions Name=TableName,Value=myTable \
    -\-statistic Sum \
    -\-threshold 240 \
    -\-comparison-operator GreaterThanOrEqualToThreshold \
    -\-period 60 \
    -\-evaluation-periods 1 \
    -\-alarm-actions arn:aws:sns:us-east-1:123456789012:capacity-alarm
```

Pruebe la alarma.

```
aws cloudwatch set-alarm-state -\-alarm-name ReadCapacityUnitsLimitAlarm -\-state-
reason "initializing" -\-state-value OK
```

```
aws cloudwatch set-alarm-state -\-alarm-name ReadCapacityUnitsLimitAlarm -\-state-
reason "initializing" -\-state-value ALARM
```

Más ejemplos de AWS CLI

En el siguiente procedimiento, se describe cómo se le notifica si tiene solicitudes que superan las cuotas de rendimiento aprovisionadas de una tabla.

1. Cree un tema de Amazon SNS `arn:aws:sns:us-east-1:123456789012:requests-exceeding-throughput`. Para obtener más información, consulte Configuración de Amazon Simple Notification Service.
2. Cree la alarma.

```
aws cloudwatch put-metric-alarm \
        -\-alarm-name ReadCapacityUnitsLimitAlarm \
        -\-alarm-description "Alarm when read capacity reaches 80% of my
 provisioned read capacity" \
        -\-namespace AWS/DynamoDB \
        -\-metric-name ConsumedReadCapacityUnits \
        -\-dimensions Name=TableName,Value=myTable \
        -\-statistic Sum \
        -\-threshold 240 \
        -\-comparison-operator GreaterThanOrEqualToThreshold \
        -\-period 60 \
        -\-evaluation-periods 1 \
        -\-alarm-actions arn:aws:sns:us-east-1:123456789012:capacity-alarm
```

3. Pruebe la alarma.

```
aws cloudwatch set-alarm-state --alarm-name RequestsExceedingThroughputAlarm --
state-reason "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name RequestsExceedingThroughputAlarm --
state-reason "initializing" --state-value ALARM
```

En el siguiente procedimiento, se describe cómo se le notifica si se producen errores del sistema.

1. Cree un tema de Amazon SNS `arn:aws:sns:us-east-1:123456789012:notify-on-system-errors`. Para obtener más información, consulte Configuración de Amazon Simple Notification Service.
2. Cree la alarma.

```
aws cloudwatch put-metric-alarm \
    --alarm-name SystemErrorsAlarm \
    --alarm-description "Alarm when system errors occur" \
    --namespace AWS/DynamoDB \
    --metric-name SystemErrors \
```

```
    --dimensions Name=TableName,Value=myTable
 Name=Operation,Value=aDynamoDBOperation \
    --statistic Sum \
    --threshold 0 \
    --comparison-operator GreaterThanThreshold \
    --period 60 \
    --unit Count \
    --evaluation-periods 1 \
    --treat-missing-data breaching \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:notify-on-system-errors
```

3. Pruebe la alarma.

```
aws cloudwatch set-alarm-state --alarm-name SystemErrorsAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name SystemErrorsAlarm --state-reason
 "initializing" --state-value ALARM
```

Registrar las operaciones de DynamoDB mediante AWS CloudTrail

DynamoDB se integra con AWS CloudTrail, un servicio que proporciona un registro de las acciones llevadas a cabo por un usuario, un rol o un servicio de AWS en DynamoDB. CloudTrail captura todas las llamadas a la API para DynamoDB como eventos. Las llamadas capturadas incluyen las llamadas desde la consola de DynamoDB y las llamadas desde el código a las operaciones de la API de DynamoDB, utilizando tanto PartiQL como la API clásica. Si crea un registro de seguimiento, puede habilitar la entrega continua de eventos de CloudTrail a un bucket de Amazon S3, incluidos los eventos de DynamoDB. Si no configura un registro de seguimiento, puede ver los eventos más recientes en la consola de CloudTrail en el Historial de eventos. Mediante la información recopilada por CloudTrail, puede determinar la solicitud que se realizó a DynamoDB, la dirección IP desde la que se realizó, quién la realizó y cuándo, etc.

Para conseguir un monitoreo y alertas robustos, también puede integrar eventos de CloudTrail con Amazon CloudWatch Logs. Para mejorar el análisis de la actividad de servicio de DynamoDB e identificar cambios en las actividades de una cuenta de AWS, puede consultar los registros de AWS CloudTrail utilizando Amazon Athena. Por ejemplo, puede ejecutar consultas que identifiquen tendencias y aislar la actividad por atributos, como el usuario o la dirección IP de origen.

Para obtener más información acerca de CloudTrail, incluso cómo configurarlo y habilitarlo, consulte la Guía del usuario de AWS CloudTrail.

Temas

- Información de DynamoDB en CloudTrail
- Descripción de las entradas del archivo de registros de DynamoDB

Información de DynamoDB en CloudTrail

CloudTrail se habilita en su cuenta de AWS cuando la crea. Cuando se produce una actividad de eventos compatible en DynamoDB, la actividad se registra en un evento de CloudTrail junto con otros eventos de servicios de AWS en Event history (Historial de eventos). Puede ver, buscar y descargar los últimos eventos de la cuenta de AWS. Para obtener más información, consulte Trabajar con el historial de eventos de CloudTrail.

Para mantener un registro continuo de eventos en su cuenta de AWS, incluyendo los eventos de DynamoDB, cree un seguimiento. Un registro de seguimiento permite a CloudTrail enviar archivos de registro a un bucket de Amazon S3. De manera predeterminada, cuando se crea un registro de seguimiento en la consola, el registro de seguimiento se aplica a todas las regiones de AWS. El registro de seguimiento registra los eventos de todas las regiones de la partición de AWS y envía los archivos de registro al bucket de Amazon S3 especificado. También es posible configurar otros servicios de AWS para analizar en profundidad y actuar en función de los datos de eventos recopilados en los registros de CloudTrail. Para más información, consulte los siguientes temas:

- Introducción a la creación de registros de seguimiento
- Servicios e integraciones compatibles con CloudTrail
- Configuración de notificaciones de Amazon SNS para CloudTrail
- Recibir archivos de registro de CloudTrail de varias regiones y Recibir archivos de registro de CloudTrail de varias cuentas

Eventos del plano de control en CloudTrail

Las siguientes acciones de la API se registran de forma predeterminada como eventos en archivos de CloudTrail:

Amazon DynamoDB

- CreateBackup
- CreateGlobalTable
- CreateTable
- DeleteBackup
- DeleteTable
- DescribeBackup
- DescribeContinuousBackups
- DescribeGlobalTable
- DescribeLimits
- DescribeTable
- DescribeTimeToLive
- ListBackups
- ListTables
- ListTagsOfResource
- ListGlobalTables
- RestoreTableFromBackup
- RestoreTableToPointInTime
- TagResource
- UntagResource
- UpdateGlobalTable
- UpdateTable
- UpdateTimeToLive
- DescribeReservedCapacity
- DescribeReservedCapacityOfferings
- PurchaseReservedCapacityOfferings
- DescribeScalableTargets
- RegisterScalableTarget

DynamoDB Streams

- DescribeStream

- ListStreams

DynamoDB Accelerator (DAX)

- CreateCluster
- CreateParameterGroup
- CreateSubnetGroup
- DecreaseReplicationFactor
- DeleteCluster
- DeleteParameterGroup
- DeleteSubnetGroup
- DescribeClusters
- DescribeDefaultParameters
- DescribeEvents
- DescribeParameterGroups
- DescribeParameters
- DescribeSubnetGroups
- IncreaseReplicationFactor
- ListTags
- RebootNode
- TagResource
- UntagResource
- UpdateCluster
- UpdateParameterGroup
- UpdateSubnetGroup

Eventos del plano de datos de DynamoDB en CloudTrail

Para habilitar el registro de las siguientes acciones de API en los archivos de CloudTrail, debe habilitar el registro de la actividad de la API del plano de datos en CloudTrail. Para obtener más información, consulte Registro de eventos de datos para seguimiento.

Los eventos del plano de datos pueden filtrarse por tipo de recurso, para tener un control pormenorizado de las llamadas a la API de DynamoDB que desea registrar y pagar de forma selectiva en CloudTrail. Por ejemplo, al especificar `AWS::DynamoDB::Stream` como tipo de recurso, puede registrar solo las llamadas a las API de flujos de DynamoDB. Para las tablas con flujos habilitados, el campo de recursos del evento del plano de datos contiene `AWS::DynamoDB::Stream` y `AWS::DynamoDB::Table`. Si lo especifica `AWS::DynamoDB::Table` como tipo de recurso, registrará tanto los eventos de la tabla de DynamoDB como los de los flujos de DynamoDB de forma predeterminada. Puede agregar un filtro adicional para excluir los eventos de flujo, si no desea que se registren los eventos de flujo. Para obtener más información, consulte DataResource en la Referencia de la API de AWS CloudTrail.

Amazon DynamoDB

- BatchExecuteStatement
- BatchGetItem
- BatchWriteItem
- DeleteItem
- ExecuteStatement
- ExecuteTransaction
- GetItem
- PutItem
- Query
- Scan
- TransactGetItems
- TransactWriteItems
- UpdateItem

> **Note**
>
> CloudTrail no registra las acciones del plano de datos de período de vida de DynamoDB

DynamoDB Streams

- GetRecords

- GetShardIterator

Descripción de las entradas del archivo de registros de DynamoDB

Un registro de seguimiento es una configuración que permite la entrega de eventos como archivos de registros en un bucket de Amazon S3 que especifique. Los archivos de registro de CloudTrail pueden contener una o varias entradas de registro. Un evento representa una solicitud específica realizada desde un origen cualquiera, y contiene información sobre la acción solicitada, la fecha y la hora de la acción, los parámetros de la solicitud, etc.

Cada entrada de registro o evento contiene información sobre quién generó la solicitud. La información de identidad del usuario le ayuda a determinar lo siguiente:

- Si la solicitud se realizó con las credenciales raíz o del usuario.
- Si la solicitud se realizó con credenciales de seguridad temporales de un rol o fue un usuario federado.
- Si la solicitud la realizó otro servicio de AWS.

> **Note**
>
> Los valores de atributo que no son clave se eliminarán de los registros de acciones de CloudTrail utilizando la API PartiQL y no aparecerán en los registros de acciones que utilicen la API clásica.

Para obtener más información, consulte Elemento userIdentity de CloudTrail.

Los siguientes ejemplos demuestran los registros de CloudTrail de estos tipos de eventos:

Amazon DynamoDB

- `UpdateTable`
- `DeleteTable`
- `CreateCluster`
- `PutItem` (exitoso)
- `UpdateItem` (sin éxito)
- `TransactWriteItems` (exitoso)

- TransactWriteItems (con TransactionCanceledException)
- ExecuteStatement
- BatchExecuteStatement

DynamoDB Streams

- GetRecords

UpdateTable

```
{
    "Records": [
        {
            "eventVersion": "1.03",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    }
                }
            },
            "eventTime": "2015-05-04T02:14:52Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "UpdateTable",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "console.aws.amazon.com",
            "requestParameters": {
```

```
            "provisionedThroughput": {
                "writeCapacityUnits": 25,
                "readCapacityUnits": 25
            }
        },
        "responseElements": {
            "tableDescription": {
                "tableName": "Music",
                "attributeDefinitions": [
                    {
                        "attributeType": "S",
                        "attributeName": "Artist"
                    },
                    {
                        "attributeType": "S",
                        "attributeName": "SongTitle"
                    }
                ],
                "itemCount": 0,
                "provisionedThroughput": {
                    "writeCapacityUnits": 10,
                    "numberOfDecreasesToday": 0,
                    "readCapacityUnits": 10,
                    "lastIncreaseDateTime": "May 3, 2015 11:34:14 PM"
                },
                "creationDateTime": "May 3, 2015 11:34:14 PM",
                "keySchema": [
                    {
                        "attributeName": "Artist",
                        "keyType": "HASH"
                    },
                    {
                        "attributeName": "SongTitle",
                        "keyType": "RANGE"
                    }
                ],
                "tableStatus": "UPDATING",
                "tableSizeBytes": 0
            }
        },
        "requestID": "AALNP0J2L244N5O15PKISJ1KUFVV4KQNSO5AEMVJF66Q9ASUAAJG",
        "eventID": "eb834e01-f168-435f-92c0-c36278378b6e",
        "eventType": "AwsApiCall",
        "apiVersion": "2012-08-10",
```

```
            "recipientAccountId": "111122223333"
        }
    ]
}
```

DeleteTable

```
{
    "Records": [
        {
            "eventVersion": "1.03",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    }
                }
            },
            "eventTime": "2015-05-04T13:38:20Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "DeleteTable",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "console.aws.amazon.com",
            "requestParameters": {
                "tableName": "Music"
            },
            "responseElements": {
                "tableDescription": {
                    "tableName": "Music",
```

```
                    "itemCount": 0,
                    "provisionedThroughput": {
                        "writeCapacityUnits": 25,
                        "numberOfDecreasesToday": 0,
                        "readCapacityUnits": 25
                    },
                    "tableStatus": "DELETING",
                    "tableSizeBytes": 0
                }
            },
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "recipientAccountId": "111122223333"
        }
    ]
}
```

CreateCluster

```
{
    "Records": [
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "userName": "bob"
            },
            "eventTime": "2019-12-17T23:17:34Z",
            "eventSource": "dax.amazonaws.com",
            "eventName": "CreateCluster",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.16.304 Python/3.6.9
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 botocore/1.13.40",
            "requestParameters": {
                "sSESpecification": {
                    "enabled": true
```

```
                },
                "clusterName": "daxcluster",
                "nodeType": "dax.r4.large",
                "replicationFactor": 3,
                "iamRoleArn": "arn:aws:iam::111122223333:role/
DAXServiceRoleForDynamoDBAccess"
            },
            "responseElements": {
                "cluster": {
                    "securityGroups": [
                        {
                            "securityGroupIdentifier": "sg-1af6e36e",
                            "status": "active"
                        }
                    ],
                    "parameterGroup": {
                        "nodeIdsToReboot": [],
                        "parameterGroupName": "default.dax1.0",
                        "parameterApplyStatus": "in-sync"
                    },
                    "clusterDiscoveryEndpoint": {
                        "port": 8111
                    },
                    "clusterArn": "arn:aws:dax:us-west-2:111122223333:cache/
daxcluster",
                    "status": "creating",
                    "subnetGroup": "default",
                    "sSEDescription": {
                        "status": "ENABLED",
                        "kMSMasterKeyArn": "arn:aws:kms:us-
west-2:111122223333:key/764898e4-adb1-46d6-a762-e2f4225b4fc4"
                    },
                    "iamRoleArn": "arn:aws:iam::111122223333:role/
DAXServiceRoleForDynamoDBAccess",
                    "clusterName": "daxcluster",
                    "activeNodes": 0,
                    "totalNodes": 3,
                    "preferredMaintenanceWindow": "thu:13:00-thu:14:00",
                    "nodeType": "dax.r4.large"
                }
            },
            "requestID": "585adc5f-ad05-4e27-8804-70ba1315f8fd",
            "eventID": "29158945-28da-4e32-88e1-56d1b90c1a0c",
            "eventType": "AwsApiCall",
```

```
            "recipientAccountId": "111122223333"
        }
    ]
}
```

PutItem (exitoso)

```
{
    "Records": [
        {
            "eventVersion": "1.06",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "attributes": {
                        "mfaAuthenticated": "false",
                        "creationDate": "2015-05-28T18:06:01Z"
                    },
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    }
                }
            },
            "eventTime": "2019-01-19T15:41:54Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "PutItem",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
 botocore/1.10.63",
            "requestParameters": {
                "tableName": "Music",
                "key": {
                    "Artist": "No One You Know",
                    "SongTitle": "Scared of My Shadow"
```

```
                },
                "item": [
                    "Artist",
                    "SongTitle",
                    "AlbumTitle"
                ],
                "returnConsumedCapacity": "TOTAL"
            },
            "responseElements": null,
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

UpdateItem (sin éxito)

```
{
    "Records": [
        {
            "eventVersion": "1.07",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
```

```
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2020-09-03T22:27:15Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "UpdateItem",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
botocore/1.10.63",
            "errorCode": "ConditionalCheckFailedException",
            "errorMessage": "The conditional request failed",
            "requestParameters": {
                "tableName": "Music",
                "key": {
                    "Artist": "No One You Know",
                    "SongTitle": "Call Me Today"
                },
                "updateExpression": "SET #Y = :y, #AT = :t",
                "expressionAttributeNames": {
                    "#Y": "Year",
                    "#AT": "AlbumTitle"
                },
                "conditionExpression": "attribute_not_exists(#Y)",
                "returnConsumedCapacity": "TOTAL"
            },
            "responseElements": null,
            "requestID": "4KBNVRGD25RG1KEO9UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "a954451c-c2fc-4561-8aea-7a30ba1fdf52",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
```

```
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

`TransactWriteItems` (exitoso)

```
{
    "Records": [
        {
            "eventVersion": "1.07",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2020-09-03T21:48:12Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "TransactWriteItems",
            "awsRegion": "us-west-1",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.15.64 Python/2.7.16 Darwin/17.7.0
 botocore/1.10.63",
```

```
            "requestParameters": {
                "requestItems": [
                    {
                        "operation": "Put",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Today"
                        },
                        "items": [
                            "Artist",
                            "SongTitle",
                            "AlbumTitle"
                        ],
                        "conditionExpression": "#AT = :A",
                        "expressionAttributeNames": {
                            "#AT": "AlbumTitle"
                        },
                        "returnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "Update",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Tomorrow"
                        },
                        "updateExpression": "SET #AT = :newval",
                        "ConditionExpression": "attribute_not_exists(Rating)",
                        "ExpressionAttributeNames": {
                            "#AT": "AlbumTitle"
                        },
                        "returnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "Delete",
                        "TableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Yesterday"
                        },
                        "conditionExpression": "#P between :lo and :hi",
                        "expressionAttributeNames": {
                            "#P": "Price"
```

```
                        },
                        "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                    },
                    {
                        "operation": "ConditionCheck",
                        "TableName": "Music",
                        "Key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Now"
                        },
                        "ConditionExpression": "#P between :lo and :hi",
                        "ExpressionAttributeNames": {
                            "#P": "Price"
                        },
                        "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                    }
                ],
                "returnConsumedCapacity": "TOTAL",
                "returnItemCollectionMetrics": "SIZE"
            },
            "responseElements": null,
            "requestID": "45EN32OM6TQSMV2MI65O4L5TNFVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "4f1cc78b-5c94-4174-a6ad-3ee78605381c",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

TransactWriteItems (con **TransactionCanceledException**)

```
{
```

```
    "Records": [
        {
            "eventVersion": "1.06",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2019-02-01T00:42:34Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "TransactWriteItems",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.16.93 Python/3.4.7
Linux/4.9.119-0.1.ac.277.71.329.metal1.x86_64 botocore/1.12.83",
            "errorCode": "TransactionCanceledException",
            "errorMessage": "Transaction cancelled, please refer cancellation reasons
for specific reasons [ConditionalCheckFailed, None]",
            "requestParameters": {
                "requestItems": [
                    {
                        "operation": "Put",
                        "tableName": "Music",
                        "key": {
                            "Artist": "No One You Know",
                            "SongTitle": "Call Me Today"
                        },
                        "items": [
                            "Artist",
```

```
                        "SongTitle",
                        "AlbumTitle"
                    ],
                    "conditionExpression": "#AT = :A",
                    "expressionAttributeNames": {
                        "#AT": "AlbumTitle"
                    },
                    "returnValuesOnConditionCheckFailure": "ALL_OLD"
                },
                {
                    "operation": "Update",
                    "tableName": "Music",
                    "key": {
                        "Artist": "No One You Know",
                        "SongTitle": "Call Me Tomorrow"
                    },
                    "updateExpression": "SET #AT = :newval",
                    "ConditionExpression": "attribute_not_exists(Rating)",
                    "ExpressionAttributeNames": {
                        "#AT": "AlbumTitle"
                    },
                    "returnValuesOnConditionCheckFailure": "ALL_OLD"
                },
                {
                    "operation": "Delete",
                    "TableName": "Music",
                    "key": {
                        "Artist": "No One You Know",
                        "SongTitle": "Call Me Yesterday"
                    },
                    "conditionExpression": "#P between :lo and :hi",
                    "expressionAttributeNames": {
                        "#P": "Price"
                    },
                    "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                },
                {
                    "operation": "ConditionCheck",
                    "TableName": "Music",
                    "Key": {
                        "Artist": "No One You Know",
                        "SongTitle": "Call Me Now"
                    },
                    "ConditionExpression": "#P between :lo and :hi",
```

```
                        "ExpressionAttributeNames": {
                            "#P": "Price"
                        },
                        "ReturnValuesOnConditionCheckFailure": "ALL_OLD"
                    }
                ],
                "returnConsumedCapacity": "TOTAL",
                "returnItemCollectionMetrics": "SIZE"
            },
            "responseElements": null,
            "requestID": "A0GTQEKLBB9VD8E05REA5A3E1VVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "43e437b5-908a-46af-84e6-e27fffb9c5cd",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

ExecuteStatement

```
{
    "Records": [
        {
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
```

```
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-03-03T23:06:45Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "ExecuteStatement",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.7 Python/3.6.13
 Linux/4.9.230-0.1.ac.223.84.332.metal1.x86_64 botocore/1.20.7",
            "requestParameters": {
                "statement": "SELECT * FROM Music WHERE Artist = 'No One You Know' AND
 SongTitle = 'Call Me Today' AND nonKeyAttr = ***(Redacted)"
            },
            "responseElements": null,
            "requestID": "V7G2KCSFLP83ORB7MMFG6RIAD3VV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "0b5c4779-e169-4227-a1de-6ed01dd18ac7",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

BatchExecuteStatement


```
{
    "Records": [
        {
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-03-03T23:24:48Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "BatchExecuteStatement",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.7 Python/3.6.13
Linux/4.9.230-0.1.ac.223.84.332.metal1.x86_64 botocore/1.20.7",
            "requestParameters": {
                "requestItems": [
                    {
                        "statement": "UPDATE Music SET Album = ***(Redacted) WHERE
Artist = 'No One You Know' AND SongTitle = 'Call Me Today'"
                    },
                    {
                        "statement": "INSERT INTO Music VALUE {'Artist' :
***(Redacted), 'SongTitle' : ***(Redacted), 'Album' : ***(Redacted)}"
                    }
```

```
                ]
            },
            "responseElements": null,
            "requestID": "23PE7ED291UD65P9SMS6TISNVBVV4KQNSO5AEMVJF66Q9ASUAAJG",
            "eventID": "f863f966-b741-4c36-b15e-f867e829035a",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
            "eventCategory": "Data"
        }
    ]
}
```

GetRecords

```
{
    "Records": [
        {
            "eventVersion": "1.08",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "AKIAIOSFODNN7EXAMPLE:bob",
                "arn": "arn:aws:sts::111122223333:assumed-role/users/bob",
                "accountId": "111122223333",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "AKIAI44QH8DHBEXAMPLE",
                        "arn": "arn:aws:iam::444455556666:role/admin-role",
                        "accountId": "444455556666",
                        "userName": "bob"
                    },
                    "attributes": {
```

```
                        "creationDate": "2020-09-03T22:14:13Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2021-04-15T04:15:02Z",
            "eventSource": "dynamodb.amazonaws.com",
            "eventName": "GetRecords",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "aws-cli/1.19.50 Python/3.6.13
 Linux/4.9.230-0.1.ac.224.84.332.metal1.x86_64 botocore/1.20.50",
            "requestParameters": {
                "shardIterator": "arn:aws:dynamodb:us-west-2:123456789012:table/
Music/stream/2021-04-15T04:02:47.428|1|AAAAAAAAAAH7HF3xwDQHBrvk2UBZ1PKh8bX3F
+JeH0rFwHCE7dz4VGV1ZoJ5bMxQwkmerA3wzCTL+zSseGLdSXNJP14EwrjLNvDNoZeRSJ/
n6xc3I4NYOptR4zR8d7VrjMAD6h5nR12NtxGIgJ/
dVsUpluWsHyCW3PPbKsMlJSruVRWoitRhSd3S6slEWEPB0bDC7+
+ISH5mXrCHOnvyezQKlqNshTSPZ5jWwqRj2VNSXCMTGXv9P01/
U0bpOUI2cuRTchgUpPSe3ur2sQrRj3KlbmIyCz7P
+H3CYlugafi8fQ5kipDSkESkIWS6O5ejzibWKg/3izms1eVIm/
zLFdEeihCYJ7G8fpHUSLX5JAk3ab68aUXGSFEZLONntgNIhQkcMo00/
mJlaIgkEdBUyqvZO1vtKUBH5YonIrZqSUhv8Coc+mh24vOg1YI+SPIXlr
+Lnl54BG6AjrmaScjHACVXoPDxPsXSJXC4c9HjoC3YSskCPV7uWi0f65/
n7JAT3cskcX2ISaLHwYzJPaMBSftxOgeRLm3BnisL32nT8uTj2gF/
PUrEjdyoqTX7EerQpcaekXmOgay5Kh8n4T2uPdM83f356vRpar/
DDp8pLFD0ddb6Yvz7zU2zGdAvTod3IScC1GpTqcjRxaMhlBVZy1TnI9Cs
+7fXMdUF6xYScjR2725icFBNLojSFVDmsfHabXaCEpmeuXZsLbp5CjcPAHa66R8mQ5tSoFjrzOEzeB4uconEXAMPLE=="
            },
            "responseElements": null,
            "requestID": "1M0U1Q80P4LDPT7A7N1A758N2VVV4KQNSO5AEMVJF66Q9EXAMPLE",
            "eventID": "09a634f2-da7d-4c9e-a259-54aceexample",
            "readOnly": true,
            "resources": [
                {
                    "accountId": "111122223333",
                    "type": "AWS::DynamoDB::Table",
                    "ARN": "arn:aws:dynamodb:us-west-2:123456789012:table/Music"
                }
            ],
            "eventType": "AwsApiCall",
            "apiVersion": "2012-08-10",
            "managementEvent": false,
            "recipientAccountId": "111122223333",
```

```
            "eventCategory": "Data"
        }
    ]
}
```

Análisis del acceso a los datos mediante CloudWatch contributor insights for DynamoDB

Amazon CloudWatch Contributor Insights for Amazon DynamoDB es una herramienta de diagnóstico que permite identificar de un vistazo las claves de acceso más frecuente y sometidas a más limitaciones controladas de una tabla o índice. Esta herramienta utiliza CloudWatch Contributor Insights.

Cuando se habilita CloudWatch Contributor Insights for DynamoDB en una tabla o un índice secundario global, se muestran los elementos a los que más se accede o sometidos a más limitaciones controladas de esos recursos.

> **Note**
>
> Contributor Insights for DynamoDB está sujeto a cargos de CloudWatch. Para obtener más información sobre los precios, consulte Precios de Amazon CloudWatch.

Temas

- Como funciona CloudWatch Contributor Insights for DynamoDB
- Introducción a Información de colaboradores de CloudWatch para DynamoDB
- Uso de IAM con CloudWatch Contributor Insights for DynamoDB

Como funciona CloudWatch Contributor Insights for DynamoDB

Amazon DynamoDB se integra con Información de colaboradores de Amazon CloudWatch para proporcionar información sobre los elementos a los que más se accede y que están sometidos a más limitaciones de una tabla o un índice secundario global. DynamoDB le da esta información a través de las reglas, los informes y los gráficos de datos de informes de Información de colaboradores de CloudWatch

Información de colaboradores de CloudWatch para DynamoDB se ha diseñado para que no tenga impacto en el rendimiento de la tabla de DynamoDB.

Para obtener más información sobre Información de colaboradores de CloudWatch, consulte Using Contributor Insights to analyze high-cardinality data en la Guía del usuario de Amazon CloudWatch.

En las secciones siguientes se describen los conceptos y comportamientos básicos de CloudWatch Contributor Insights for DynamoDB.

Temas

- Modos de Información·de·colaboradores·de·CloudWatch para DynamoDB
- Reglas de CloudWatch Contributor Insights for DynamoDB
- Descripción de los gráficos de CloudWatch Contributor Insights for DynamoDB
- Interacciones con otras características de DynamoDB
- Facturación de CloudWatch Contributor Insights for DynamoDB

Modos de Información·de·colaboradores·de·CloudWatch para DynamoDB

Información de colaboradores de CloudWatch para DynamoDB ofrece dos modos distintos para satisfacer diferentes necesidades de supervisión.

Modo de claves limitadas

Este modo se centra exclusivamente en las solicitudes limitadas y solo procesa los eventos cuando se produce una limitación. Proporciona información sobre los problemas de rendimiento sin la sobrecarga de realizar un seguimiento de todos los patrones de acceso. En este modo, DynamoDB realiza un seguimiento solo de los:

- Elementos más limitados: elementos que experimentan más eventos de limitación

Este modo es ideal cuando:

- Su principal preocupación es identificar y resolver problemas de limitación
- Desea mantener Información de colaboradores habilitado de forma continua para detectar la limitación en tiempo real
- Desea un enfoque optimizado en costos para supervisar los problemas de limitación

> **Note**
>
> El modo de claves limitadas procesa los eventos solo cuando se produce una limitación, lo que hace que sea rentable para la supervisión continua. Este enfoque específico le permite dejar la característica habilitada permanentemente con un impacto económico mínimo y, al mismo tiempo, ofrecer una visibilidad inmediata de los problemas de limitación a medida que se producen.

Si la tabla no experimenta ninguna limitación, no verá ningún dato en los gráficos de Información de colaboradores, lo que indica un buen rendimiento. Cuando se detecta una limitación, los gráficos generados le ayudan a identificar patrones de acceso específicos que causan problemas de rendimiento. Esta información puede ayudarle a implementar estrategias para abordar los patrones de acceso no uniformes.

Para obtener estrategias de supervisión integrales, puede integrar esta información de limitación con otras métricas de CloudWatch para crear paneles unificados que correlacionen los eventos de limitación con el rendimiento general de la tabla.

Modo de claves a las que se ha accedido y limitadas

Este modo proporciona una supervisión integral de los elementos a los que se ha accedido y los que están limitados. En este modo, DynamoDB realiza un seguimiento de los:

- Elementos a los que más se accede: elementos que consumen la mayor cantidad de capacidad de lectura y escritura
- Elementos más limitados: elementos que experimentan más eventos de limitación

Este modo es ideal cuando necesita tener una visibilidad completa de los patrones de acceso de la tabla y quiere entender los elementos con mucho tráfico y los problemas de limitación.

Cambio entre modos

Puede cambiar de modo en cualquier momento mediante la consola de DynamoDB, la AWS CLI o las API. Al cambiar de modos:

- Las reglas existentes de CloudWatch se actualizan para que coincidan con el nuevo modo
- Las reglas clave de CloudWatch limitadas permanecen intactas, mantienen los datos históricos continuos para las métricas de limitación:

  - Al cambiar del modo de claves limitadas al modo de claves a las que se ha accedido y limitadas, se preservan las reglas de las claves limitadas existentes y se crean nuevas reglas de claves a las que se ha accedido
  - Al cambiar del modo de claves a las que se ha accedido y limitadas al modo de claves limitadas, solo se preservan las reglas de las claves limitadas y se eliminan las reglas de las claves a las que se ha accedido
- La facturación se ajusta inmediatamente para reflejar el procesamiento de eventos del nuevo modo

Reglas de CloudWatch Contributor Insights for DynamoDB

Cuando se habilita Información·de·colaboradores·de·CloudWatch para DynamoDB en una tabla o un índice secundario global, DynamoDB crea reglas automáticamente en función del modo seleccionado.

> **Note**
>
> Al habilitar Contributor Insights en la tabla de DynamoDB, se seguirán aplicando los límites de las reglas de Contributor Insights. Para obtener más información, consulte CloudWatch Service Quotas.

Reglas para el modo de claves a las que se ha accedido y limitadas

En el modo de claves a las que se ha accedido y limitadas, DynamoDB crea las siguientes reglas:

- Elementos a los que más se accede (claves de partición): identifica las claves de partición de los elementos a los que más se accede de la tabla o del índice secundario global.

  Formato de nombre de regla de CloudWatch: `DynamoDBContributorInsights-PKC-[resource_name]-[creationtimestamp]`
- Claves sometidas a más limitaciones (claves de partición): identifica las claves de partición de los elementos sometidos a más limitaciones de la tabla o del índice secundario global.

  Formato de nombre de regla de CloudWatch: `DynamoDBContributorInsights-PKT-[resource_name]-[creationtimestamp]`

Si la tabla o el índice secundario global tiene claves de clasificación, DynamoDB también crea las siguientes reglas específicas para las claves de clasificación:

- Most accessed keys (partition and sort keys) (Claves a las que más se accede [claves de partición y ordenación]): identifica las claves de partición y ordenación de los elementos a los que más se accede de la tabla o del índice secundario global.

  Formato de nombre de regla de CloudWatch: `DynamoDBContributorInsights-SKC-[resource_name]-[creationtimestamp]`

- Most throttled keys (partition and sort keys) (Claves sometidas a más limitaciones controladas [claves de partición y ordenación]): identifica las claves de partición y ordenación de los elementos sometidos a más limitaciones controladas de la tabla o del índice secundario global.

  Formato de nombre de regla de CloudWatch: `DynamoDBContributorInsights-SKT-[resource_name]-[creationtimestamp]`

Reglas para el modo de claves limitadas

En el modo de claves limitadas, DynamoDB crea solo las reglas relacionadas con la limitación:

- Most throttled keys (partition key) (Claves sometidas a más limitaciones controladas [clave de partición]): identifica las claves de partición de los elementos sometidos a más limitaciones controladas de la tabla o del índice secundario global.

  Formato de nombre de regla de CloudWatch: `DynamoDBContributorInsights-PKT-[resource_name]-[creationtimestamp]`

Si la tabla o el índice secundario global tiene claves de clasificación, DynamoDB también crea:

- Most throttled keys (partition and sort keys) (Claves sometidas a más limitaciones controladas [claves de partición y ordenación]): identifica las claves de partición y ordenación de los elementos sometidos a más limitaciones controladas de la tabla o del índice secundario global.

  Formato de nombre de regla de CloudWatch: `DynamoDBContributorInsights-SKT-[resource_name]-[creationtimestamp]`

Este enfoque específico reduce la cantidad de reglas activas y el volumen de eventos procesados para diagnosticar mejor los eventos de limitación.

> **Note**
>
> - Cuando usa la consola de CloudWatch o las API para ver Información de colaboradores de CloudWatch para DynamoDB, solo ve las reglas correspondientes al modo seleccionado.
> - No puede usar las API o la consola de CloudWatch para modificar o eliminar directamente las reglas creadas por CloudWatch Contributor Insights for DynamoDB. Al desactivar CloudWatch Contributor Insights for DynamoDB en una tabla o un índice secundario global, se eliminan automáticamente las reglas creadas para esa tabla o índice secundario global.
> - Cuando se usa la operación GetInsightRuleReport con reglas de CloudWatch Contributor Insights creadas por DynamoDB, solo `MaxContributorValue` y `Maximum` devuelven estadísticas útiles. Las demás estadísticas de esta lista no devuelven valores significativos.
> - CloudWatch Contributor Insights para DynamoDB tiene un límite de 25 colaboradores. Si se solicitan más de 25 colaboradores, se producirá un error.

Puede crear alarmas de CloudWatch mediante las reglasde CloudWatch Contributor Insights para DynamoDB. Esto le permite recibir una notificación cuando cualquier artículo supere o cumpla un umbral específico para `ConsumedThroughputUnits` o `ThrottleCount`. Para obtener más información, consulte Configuración de una alarma en los datos de métricas de Información de colaboradores.

Descripción de los gráficos de CloudWatch Contributor Insights for DynamoDB

Información·de·colaboradores·de·CloudWatch para DynamoDB muestra tipos de gráficos diferentes en las consolas de DynamoDB y CloudWatch en función del modo seleccionado.

Disponibilidad del gráfico por modo

Los gráficos que se muestran dependen del modo de Contributor Insights que haya seleccionado.

- El modo de claves a las que se ha accedido y limitadas muestra los gráficos de los elementos a los que más se ha accedido y los elementos más limitados
- El modo de claves limitadas muestra solo los gráficos de los elementos más limitados

Most Accessed Items (Elementos más consultados)

Este gráfico solo está disponible en el modo de claves a las que se ha accedido y limitadas. Use este gráfico para identificar los elementos a los que se accede con más frecuencia en la tabla o el índice secundario global. El gráfico muestra los valores de `ConsumedThroughputUnits` en el eje Y y el tiempo en el eje X. Cada una de las N claves principales se muestra con su propio color y la leyenda aparece debajo del eje X.

DynamoDB mide la frecuencia de acceso a las claves mediante `ConsumedThroughputUnits`, que mide el tráfico combinado de lectura y de escritura. `ConsumedThroughputUnits` se define como sigue:

- Aprovisionadas: (3 x unidades de capacidad de escritura consumidas) + Unidades de capacidad de lectura consumidas.
- En diferido: (3 x unidades de solicitud de escritura) + unidades de solicitud de lectura.

En la consola de DynamoDB, cada punto de datos del gráfico representa el máximo de `ConsumedThroughputUnits` durante un periodo de un minuto. Por ejemplo, un valor de 180 000 `ConsumedThroughputUnits` en el gráfico indica que se ha accedido continuamente a ese elemento al máximo rendimiento por elemento de 1000 unidades de solicitud de escritura o 3000 unidades de solicitud de lectura a lo largo de 60 segundos durante ese periodo de un minuto (3000 x 60 segundos). Dicho de otro modo, los valores del gráfico representan el minuto con el máximo tráfico de cada periodo de un minuto. Puede cambiar la granularidad de tiempo de la métrica `ConsumedThroughputUnits` (por ejemplo, para ver métricas de 5 minutos en lugar de 1 minuto) en la consola de CloudWatch.

Si aparecen varias líneas muy agrupadas sin valores atípicos evidentes, indica que la carga de trabajo está relativamente equilibrada para todos los elementos durante ese espacio de tiempo determinado. Si aparecen puntos aislados en el gráfico en lugar de líneas conectadas, indican un elemento al que se ha accedido con frecuencia únicamente durante un breve periodo.

Si la tabla o el índice secundario global tiene claves de clasificación, DynamoDB crea dos gráficos: uno para las claves de partición a las que más se accede y otro para los pares de clave de partición y clave de clasificación de acceso más frecuente. Puede ver el tráfico por claves de partición en el gráfico de solo clave de partición. Puede ver el tráfico por elemento en los gráficos de clave de clasificación + partición.

Most Throttled Items (Elementos sometidos a más limitaciones controladas)

Este gráfico está disponible en ambos modos. Use este gráfico para identificar los elementos sometidos a más limitaciones controladas en la tabla o el índice secundario global. El gráfico muestra los valores de `ThrottleCount` en el eje Y y el tiempo en el eje X. Cada una de las N claves principales se muestra con su propio color y la leyenda aparece debajo del eje X.

DynamoDB mide la frecuencia de las limitaciones controladas mediante `ThrottleCount`, que es la cantidad de excepciones `ProvisionedThroughputExceededException` y `ThrottlingException` y de errores `RequestLimitExceeded`.

No se mide la limitación controlada de la escritura causada por una capacidad de escritura insuficiente para un índice secundario global. Puede utilizar el gráfico Most Accessed Items (Elementos a los que más se accede) del índice secundario global para identificar patrones de acceso desequilibrados que pueden causar la limitación controlada de la escritura. Para obtener más información, consulte Consideraciones sobre el rendimiento aprovisionado para los índices secundarios globales.

En la consola de DynamoDB, cada punto de datos del gráfico representa el número de eventos de limitación controlada durante un período de un minuto.

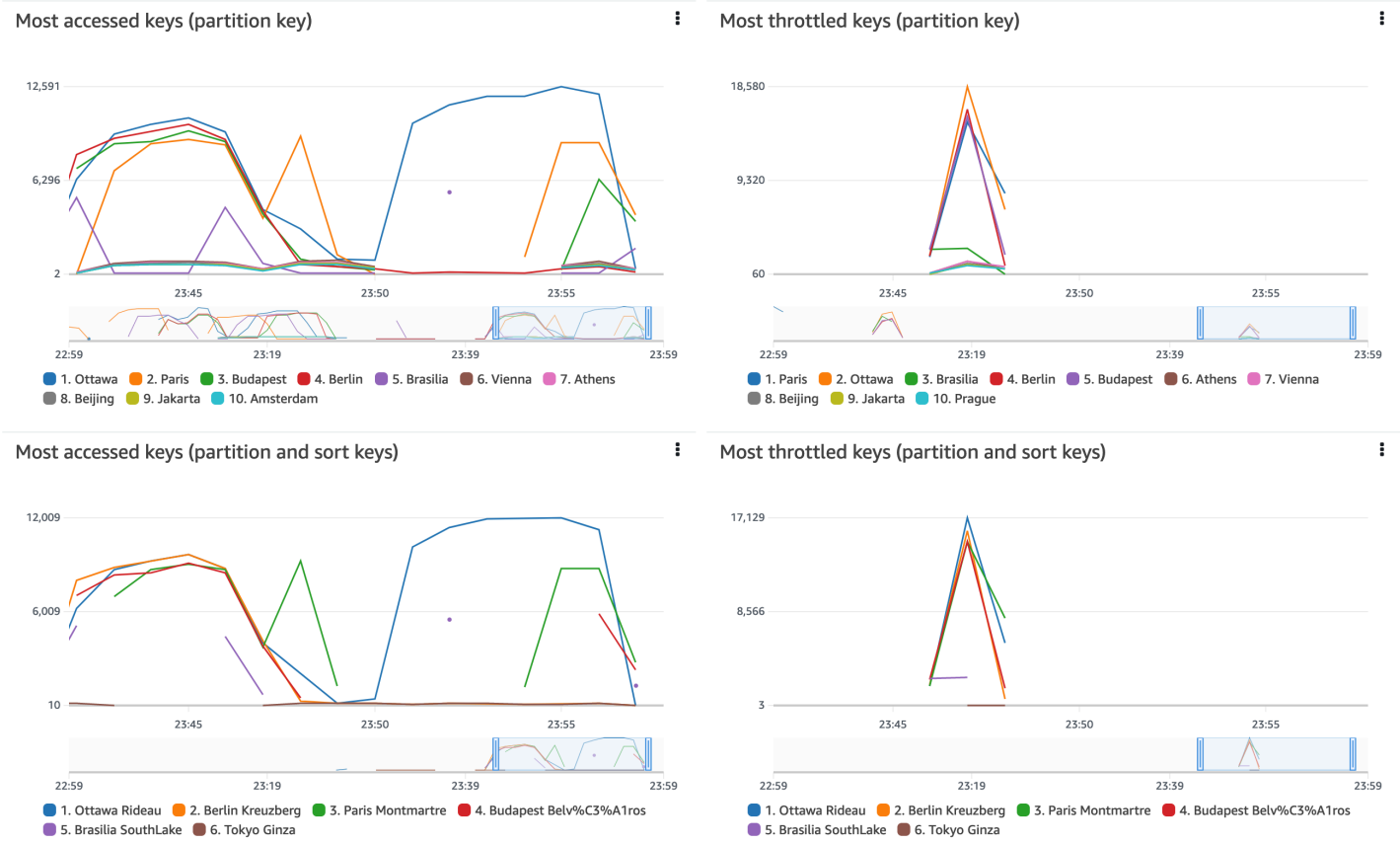
Si no aparecen datos en este gráfico, indica que las solicitudes no se están sometido a limitaciones controladas. Si ve puntos aislados en el gráfico en lugar de líneas conectadas, indican un elemento que se ha sometido a limitaciones controladas durante un breve periodo.

Si la tabla o el índice secundario global tiene claves de clasificación, DynamoDB crea dos gráficos: uno para las claves de partición sometidas a más limitaciones y otro para los pares de clave de partición y clave de clasificación sometidos a más limitaciones. Puede ver la cantidad de limitaciones por claves de partición en el gráfico de solo claves de partición, y la cantidad de limitaciones por elemento en los gráficos de partición y clave de clasificación.

> **Note**
>
> En el modo de claves limitadas, este es el único tipo de gráfico que verá. La ausencia de datos en estos gráficos indica un buen rendimiento de la tabla sin que se produzcan limitaciones.

Ejemplos de informes

En el ejemplo siguiente, se muestran los informes generados para una tabla que tiene claves de partición y claves de clasificación en los modos de claves a las que se ha accedido y limitadas. En el modo de claves limitadas, solo verá la parte de este informe relacionada con la limitación.



Interacciones con otras características de DynamoDB

En las siguientes secciones se describe cómo CloudWatch Contributor Insights for DynamoDB se comporta e interactúa con otras características de DynamoDB. Estos comportamientos se aplican a ambos modos a menos que se especifique lo contrario.

Tablas globales

CloudWatch Contributor Insights for DynamoDB monitorea las réplicas de tablas globales como tablas distintas. Es posible que los gráficos de Contributor Insights de una réplica en una región de AWS no muestren los mismos patrones que otra región. Esto se debe a que los datos de escritura se replican en todas las réplicas de una tabla global, pero cada réplica solo atiende al tráfico de lectura de una región.

Cada réplica se puede configurar con un modo de Contributor Insights diferente de forma independiente. Por ejemplo, puede usar el modo de claves a las que se ha accedido y limitadas en la región principal para una supervisión integral, mientras que puede usar el modo de claves limitadas en regiones secundarias para mantener la visibilidad de los problemas de rendimiento.

DynamoDB Accelerator (DAX)

CloudWatch Contributor Insights for DynamoDB no muestra respuestas de caché de DAX. Sólo muestra respuestas para acceder a una tabla o a un índice secundario global.

> **Note**
>
> Información de colaboradores de CloudWatch de DynamoDB no admite solicitudes de PartiQL.

Cifrado en reposo

CloudWatch Contributor Insights for DynamoDB no afecta al modo en que funciona el cifrado en DynamoDB. Los datos de clave principal que se publican en CloudWatch se cifran con la Clave propiedad de AWS. Sin embargo, DynamoDB también admite la Clave administrada de AWS y una clave administrada por el cliente.

Información de colaboradores de CloudWatch para DynamoDB muestra las claves de partición y las claves de clasificación (si es aplicable) de los elementos con limitación a los que se accede con frecuencia. Aunque Información de colaboradores de CloudWatch funciona con tablas de DynamoDB cifradas, es importante tener en cuenta que utiliza el contexto de cifrado propiedad de Amazon, que es independiente del cifrado configurado de la tabla.

Si la clave principal de la tabla de DynamoDB contiene información confidencial y las políticas de seguridad de la organización requieren un control total sobre los procesos de cifrado, habilitar Información de colaboradores de CloudWatch puede no ser adecuado.

Control de acceso detallado

CloudWatch Contributor Insights for DynamoDB no funciona de manera diferente para las tablas con control de acceso detallado (FGAC, por sus siglas en inglés). Es decir, cualquier usuario que tenga los permisos de CloudWatch apropiados puede ver las claves principales protegidas mediante FGAC en los gráficos de CloudWatch Contributor Insights.

Si la clave principal de la tabla contiene datos protegidos mediante FGAC que no desea publicar en CloudWatch, no debería habilitar Información·de·colaboradores·de·CloudWatch para DynamoDB para esa tabla.

Control de acceso

Para controlar el acceso a CloudWatch Contributor Insights for DynamoDB mediante AWS Identity and Access Management (IAM), limite los permisos del plano de control de DynamoDB y los permisos del plano de datos de CloudWatch. Para obtener más información, consulte el artículo sobre uso de IAM con CloudWatch Contributor Insights for DynamoDB.

Facturación de CloudWatch Contributor Insights for DynamoDB

Los cargos por CloudWatch Contributor Insights for DynamoDB aparecen en la sección CloudWatch de su factura mensual. Estos cargos se calculan en función del número de eventos de DynamoDB que se procesan y el modo seleccionado.

Facturación por modo

Los dos modos de Información de colaboradores tienen características de facturación diferentes.

- Facturación de modo de claves a las que se ha accedido y limitadas: en este modo, cada elemento que se escribe o lee a través de una operación de plano de datos representa un evento, independientemente de si la solicitud se realiza correctamente o se limita. Si una tabla o índice secundario global incluye claves de clasificación, cada elemento leído o escrito representa dos eventos. Esto se debe a que DynamoDB identifica a los principales contribuyentes de series temporales independientes: una para claves de particiones solamente y otra para pares de claves de partición y clasificación.
- Facturación de modo de claves limitadas: en este modo, solo las solicitudes limitadas generan eventos facturables. Los eventos solo se generan cuando las solicitudes generan errores `ProvisionedThroughputExceededException`, `ThrottlingException` o `RequestLimitExceeded`. Si una tabla o un índice secundario global incluye claves de clasificación, cada elemento limitado representa dos eventos (el seguimiento de las claves de partición y el seguimiento de la clave de partición + clasificación).

Ejemplos de facturación

Por ejemplo, supongamos que la aplicación realiza las siguientes operaciones de DynamoDB: una operación `GetItem`, una operación `PutItem` y una operación `BatchWriteItem` que pone cinco

elementos. Suponga también que la operación `PutItem` está limitada, pero que todas las demás operaciones se realizan correctamente.

- Modo de claves a las que se ha accedido y limitadas
  - Si la tabla o el índice secundario global solo tiene claves de partición, esto dará como resultado 7 eventos (1 para `GetItem`, 1 para `PutItem` y 5 para `BatchWriteItem`).
  - Si la tabla o el índice secundario global tiene claves de partición y claves de clasificación, esto dará como resultado 14 eventos (2 para `GetItem`, 2 para `PutItem` y 10 para `BatchWriteItem`).
- Modo de claves limitadas
  - Si la tabla o el índice secundario global solo tiene claves de partición, esto dará como resultado 1 evento (solo para `PutItem` limitado).
  - Si la tabla o el índice secundario global tiene claves de partición y claves de clasificación, esto dará como resultado 2 eventos (2 para el `PutItem` limitado).

  Las operaciones correctas `GetItem` y `BatchWriteItem` no generan eventos en el modo de claves limitadas.

Factores de facturación comunes

Una operación de `Query` siempre dará como resultado un evento, independientemente del modo o número de elementos devueltos.

A diferencia de otras características de DynamoDB, la facturación de CloudWatch Contributor Insights for DynamoDB no varía en función de lo siguiente:

- El modo de capacidad (aprovisionada frente a bajo demanda)
- Si realiza solicitudes de lectura o escritura
- El tamaño (KB) de los elementos leídos o escritos

Introducción a Información de colaboradores de CloudWatch para DynamoDB

En esta sección, se describe cómo habilitar y usar Información de colaboradores de Amazon CloudWatch en diferentes modos para cumplir las necesidades de supervisión mediante la consola de Amazon DynamoDB o la AWS Command Line Interface (AWS CLI).

En los siguientes ejemplos, usará la tabla de DynamoDB que se define en el tutorial de Introducción a DynamoDB.

Temas

- Elección de un modo de Contributor Insights
- Uso de Información de colaboradores (consola)
- Uso de Contributor Insights (AWS CLI)

Elección de un modo de Contributor Insights

Antes de habilitar Contributor Insights, debe comprender los dos modos disponibles. Revise la comparación de modos para seleccionar la opción que mejor se adapte a sus requisitos específicos.

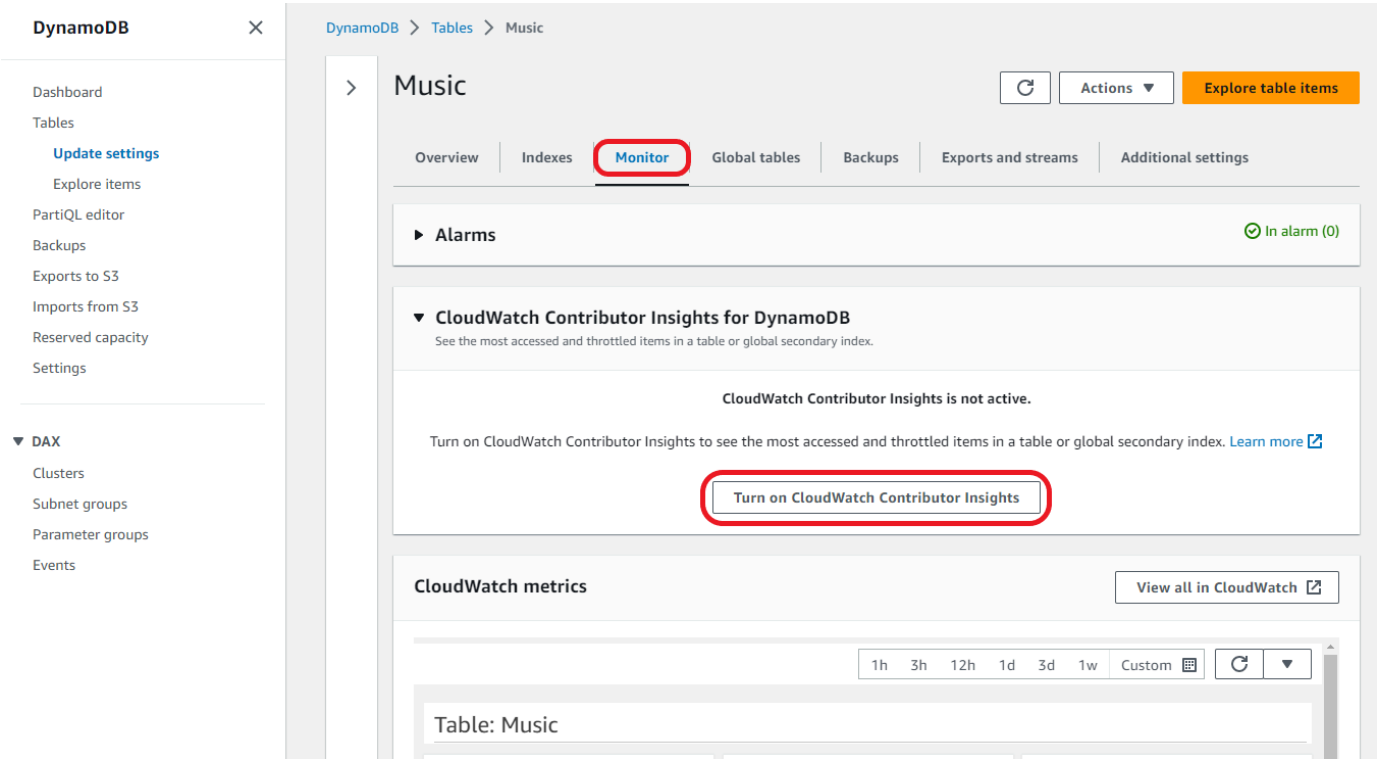
Aspecto	Modo de claves a las que se ha accedido y limitadas	Modo de claves limitadas
Monitores	Todas las solicitudes (satisfactorias y limitadas)	Solo solicitudes limitadas
Gráficos	Los elementos a los que más se accede + los elementos más limitados	Solo los elementos más limitados
Lo mejor para	Análisis y optimización específicos	Supervisión de la limitación
Usar cuando	Necesita una visibilidad completa de los patrones de acceso. Está realizando un análisis o una depuración a corto plazo.	Su principal preocupación es identificar y resolver problemas de limitación. Desea mantener Contributor Insights habilitado de forma continua para las alertas de limitación en tiempo real.

Uso de Información de colaboradores (consola)

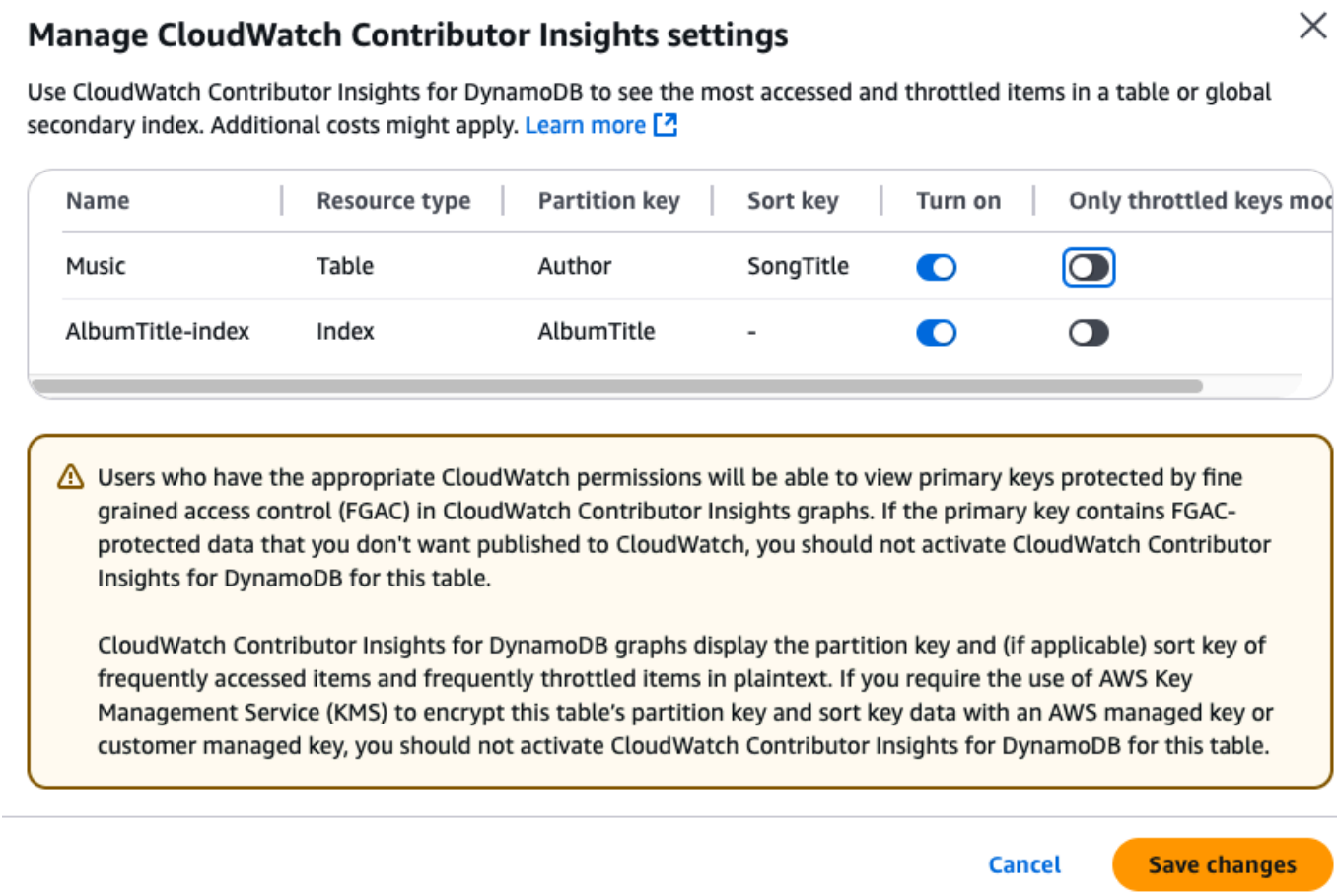
La consola proporciona una forma intuitiva de habilitar Contributor Insights y seleccionar el modo adecuado para sus necesidades de supervisión.

Uso de Información de colaboradores en la consola

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación del lado izquierdo de la consola, elija Tablas.
3. Elija la tabla `Music`.
4. Elija la pestaña Supervisar.
5. Elija Activar Información de colaboradores de CloudWatch.



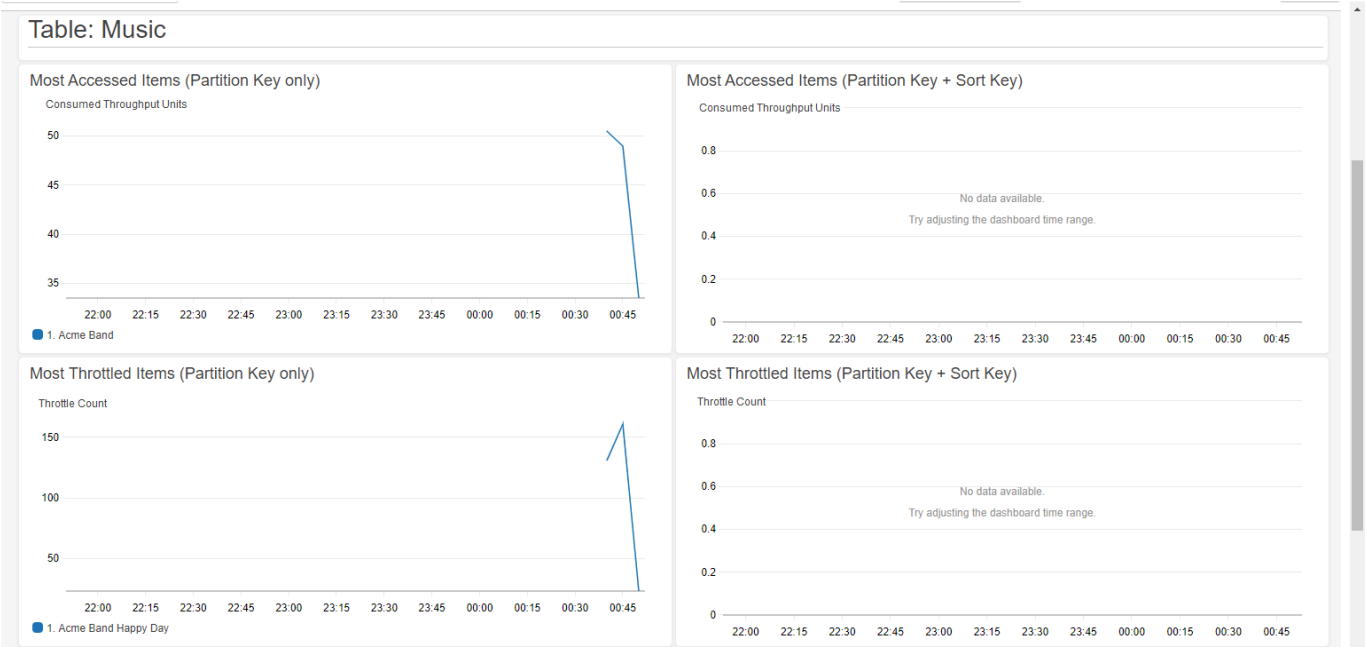
6. En el cuadro de diálogo Administrar la configuración de CloudWatch Contributor Insights, active la opción para la tabla base de `Music` y el índice secundario global `AlbumTitle-index`.
7. Deje el interruptor de Solo modo de claves limitadas en la posición de desactivado para ambos y, a continuación, elija Guardar cambios.



Esto habilita el modo predeterminado de claves a las que se ha accedido y limitadas para la tabla y GSI, lo que permite supervisar los elementos a los que se ha accedido y limitados. Cambiar el interruptor de Solo modo de claves limitadas a la posición de encendido habilitaría el modo de claves limitadas.

Si la operación no se realiza correctamente, consulte FailureException en DescribeContributorInsights en la Referencia de la API de Amazon DynamoDB para conocer los posibles motivos.

8. Los gráficos de CloudWatch Contributor Insights son ahora visibles en la pestaña Supervisar para la tabla de `Music`. Dado que ha habilitado el modo de claves a las que se ha accedido y limitadas, verá gráficos de elementos a los que se ha accedido y limitados.



Cambio entre modos

Puede cambiar entre modos en cualquier momento sin desactivar Contributor Insights.

Cambio de los modos de Contributor Insights

1. En la pestaña Supervisar de la tabla, elija Administrar CloudWatch Contributor Insights.
2. En el cuadro de diálogo Administrar la configuración de Contributor Insights, para cada tabla base o GSI:
   - Active o desactive el modo solo claves limitadas para habilitar el modo claves limitadas o volver al modo predeterminado claves a las que se ha accedido y limitadas.
   - Desactive la opción Activar para desactivar CloudWatch Contributor Insight para una tabla o un GSI.
3. Seleccione Save changes (Guardar cambios).

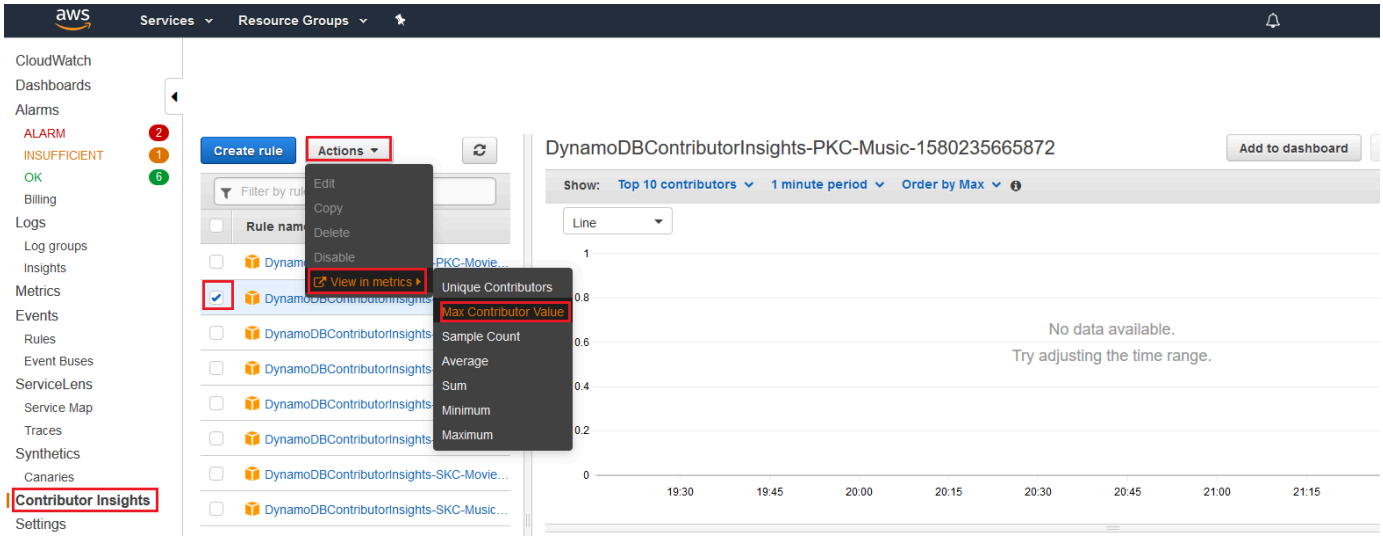
   Una vez completado, los gráficos reflejarán el modo nuevo.

Creación de alarmas de CloudWatch

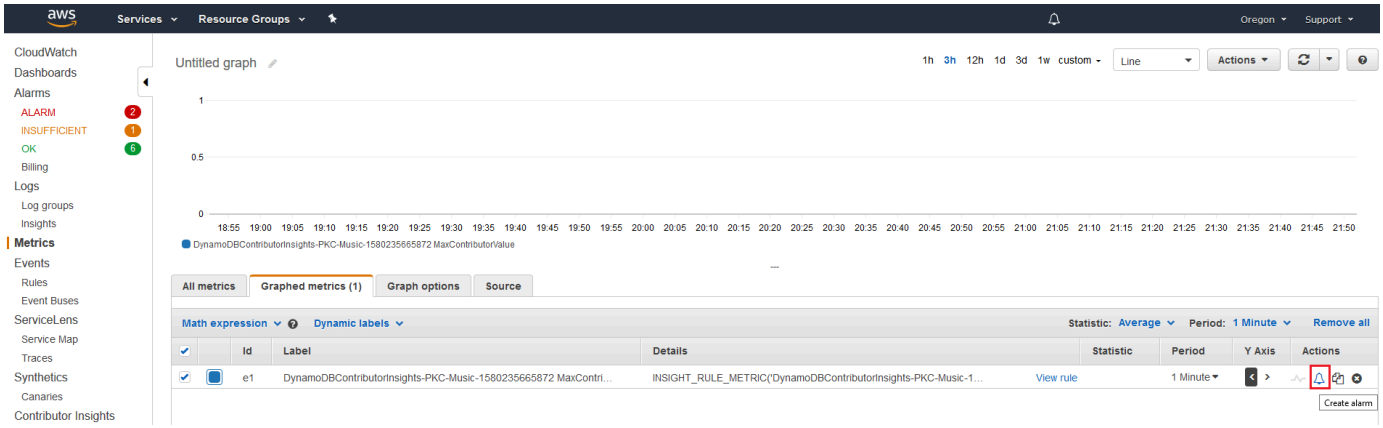
Siga estos pasos para crear una alarma de CloudWatch y recibir una notificación cuando cualquier clave de partición consume más de 50 000 ConsumedThroughputUnits o experimenta limitaciones.

1. Inicie sesión en la Consola de administración de AWS y abra la consola de CloudWatch en https://console.aws.amazon.com/cloudwatch/.
2. En el panel de navegación del lado izquierdo de la consola, seleccione Contributor Insights.
3. Elija la regla adecuada en función del modo y de lo que desee supervisar:
   - Para la supervisión de los elementos a los que se ha accedido (solo en el modo de claves a las que se ha accedido y limitadas): elija DynamoDBContributorInsights-PKC-Music
   - Para la supervisión de elementos limitados (ambos modos): elija DynamoDBContributorInsights-PKT-Music
4. Seleccione el menú desplegable Acciones.
5. Elija View in metrics (Ver en métricas).
6. Seleccione Valor máximo de colaborador.

> **Note**
>
> Solo `Max Contributor Value` y `Maximum` producen estadísticas útiles. Las demás estadísticas de esta lista no devuelven valores significativos.

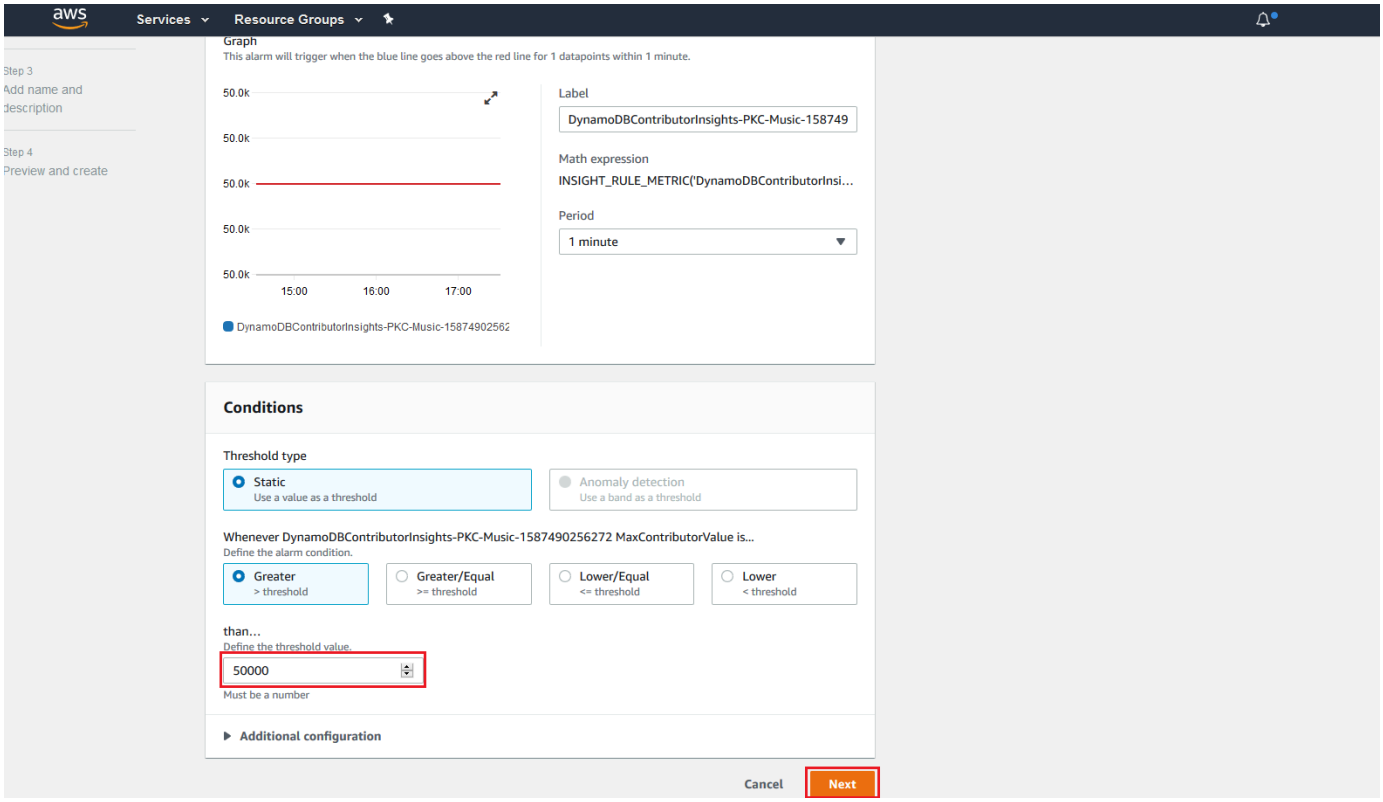


7. En la columna Acciones seleccione Crear alarma.



8. Ingrese un valor de limitación adecuado y elija Siguiente:

   - Para los elementos a los que se ha accedido (reglas de PKC): ingrese 50 000 para `ConsumedThroughputUnits`
   - Para los elementos limitados (reglas de PKT): ingrese 1 para `ThrottleCount` para recibir una alerta en caso de que se produzca una limitación



9. Consulte Uso de alarmas de Amazon CloudWatch para obtener información detallada sobre cómo configurar la notificación de la alarma.

Uso de Contributor Insights (AWS CLI)

La AWS CLI proporciona acceso programático a Contributor Insights y es totalmente compatible con ambos modos. Puede especificar el modo al habilitar Contributor Insights o cambiar de modo más adelante.

Operaciones básicas con el modo predeterminado

Uso de Contributor Insights con la configuración predeterminada

1. Habilite CloudWatch Contributor Insights para DynamoDB en la tabla base de `Music` con el modo de claves a las que se ha accedido y limitadas. Dado que `ACCESSED_AND_THROTTLED_KEYS` es el modo predeterminado, puede omitir el parámetro `--contributor-insights-mode=ACCESSED_AND_THROTTLED_KEYS`.

   ```
   aws dynamodb update-contributor-insights \
                       --table-name Music \
                       --contributor-insights-action=ENABLE
   ```

2. Habilite Contributor Insights for DynamoDB en el índice secundario global `AlbumTitle-index`.

   ```
   aws dynamodb update-contributor-insights \
                       --table-name Music \
                       --index-name AlbumTitle-index \
                       --contributor-insights-action=ENABLE
   ```

3. Obtenga el estado y las reglas de la tabla `Music` y todos sus índices.

   ```
   aws dynamodb describe-contributor-insights
                       --table-name Music
   ```

   La respuesta incluirá el campo `ContributorInsightsMode` que muestra `ACCESSED_AND_THROTTLED_KEYS`.

4. Muestre el estado de la tabla `Music` y todos sus índices.

   ```
   aws dynamodb list-contributor-insights --table-name Music
   ```

Habilitación del modo de claves limitadas

Habilitación de Contributor Insights en el modo de claves limitadas

1. Habilite CloudWatch Contributor Insights para DynamoDB en la tabla base de `Music` con el modo de claves limitadas.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

2. Habilite Contributor Insights en el modo de claves limitadas para el índice secundario global `AlbumTitle-index`.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --index-name AlbumTitle-index \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

3. Compruebe el modo describiendo la configuración de Contributor Insights.

```
aws dynamodb describe-contributor-insights --table-name Music
```

   La respuesta mostrará `ContributorInsightsMode` como `THROTTLED_KEYS` y menos reglas en comparación con el modo predeterminado.

Cambio entre modos

Cambio de los modos de Contributor Insights

1. Cambie del modo de claves limitadas al modo de claves a las que se ha accedido y limitadas.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=ACCESSED_AND_THROTTLED_KEYS
```

2. Cambie del modo de claves a las que se ha accedido y limitadas al modo de claves limitadas.

```
aws dynamodb update-contributor-insights \
    --table-name Music \
    --contributor-insights-action=ENABLE \
    --contributor-insights-mode=THROTTLED_KEYS
```

3. Compruebe el estado durante la transición.

```
aws dynamodb describe-contributor-insights --table-name Music
```

Durante el cambio de modo, `ContributorInsightsStatus` se mostrará como `ENABLING`. Una vez completado, se mostrará como `ENABLED` con el nuevo modo.

Administración de Contributor Insights

Administración de la configuración de Contributor Insights

1. Desactive CloudWatch Contributor Insights for DynamoDB en el índice secundario global `AlbumTitle-index`.

```
aws dynamodb update-contributor-insights \
                    --table-name Music --index-name AlbumTitle-index \
                    --contributor-insights-action=DISABLE
```

2. Muestre todas las configuraciones de Contributor Insights en la cuenta.

```
aws dynamodb list-contributor-insights
```

Aquí se muestran todas las tablas e índices con Contributor Insights habilitado, junto con sus modos.

3. Obtenga información detallada sobre una configuración específica.

```
aws dynamodb describe-contributor-insights \
                    --table-name Music \
                    --index-name AlbumTitle-index
```

Respuestas de ejemplo

Esto son respuestas de ejemplo que muestran las diferencias entre los modos:

Respuesta del modo de claves a las que se ha accedido y limitadas

```
{
    "TableName": "Music",
    "ContributorInsightsRuleList": [
        "DynamoDBContributorInsights-PKC-Music-1234567890123",
        "DynamoDBContributorInsights-PKT-Music-1234567890123",
        "DynamoDBContributorInsights-SKC-Music-1234567890123",
        "DynamoDBContributorInsights-SKT-Music-1234567890123"
    ],
    "ContributorInsightsStatus": "ENABLED",
    "ContributorInsightsMode": "ACCESSED_AND_THROTTLED_KEYS",
    "LastUpdateDateTime": "2024-01-15T10:30:00.000Z"
}
```

Respuesta del modo de claves limitadas

```
{
    "TableName": "Music",
    "ContributorInsightsRuleList": [
        "DynamoDBContributorInsights-PKT-Music-1234567890123",
        "DynamoDBContributorInsights-SKT-Music-1234567890123"
    ],
    "ContributorInsightsStatus": "ENABLED",
    "ContributorInsightsMode": "THROTTLED_KEYS",
    "LastUpdateDateTime": "2024-01-15T10:35:00.000Z"
}
```

Observe que el modo de claves limitadas tiene menos reglas (solo PKT y SKT), lo que corresponde a una supervisión más centrada.

Uso de IAM con CloudWatch Contributor Insights for DynamoDB

La primera vez que habilita Amazon CloudWatch Contributor Insights for Amazon DynamoDB, DynamoDB le crea automáticamente un rol vinculado a servicio de AWS Identity and Access Management (IAM). Este rol, `AWSServiceRoleForDynamoDBCloudWatchContributorInsights`, permite que DynamoDB administre las reglas de CloudWatch Contributor Insights por usted. No elimine este rol vinculado a un servicio. Si lo elimina, todas las reglas administradas dejarán de limpiarse cuando se elimine la tabla o el índice secundario global.

Para obtener más información acerca los roles vinculados a servicios, consulte Uso de roles vinculados a servicios en la Guía del usuario de IAM.

Los siguientes permisos son necesarios:

- Para habilitar o desactivar CloudWatch Contributor Insights for DynamoDB, debe tener el permiso `dynamodb:UpdateContributorInsights` para la tabla o el índice.
- Para ver los gráficos de CloudWatch Contributor Insights for DynamoDB, debe tener el permiso `cloudwatch:GetInsightRuleReport`.
- Si desea describir CloudWatch Contributor Insights for DynamoDB en una determinada tabla o índice de DynamoDB, debe tener el permiso `dynamodb:DescribeContributorInsights`.
- Para enumerar los estados de CloudWatch Contributor Insights for DynamoDB para cada tabla e índice secundario global, debe tener el permiso `dynamodb:ListContributorInsights`.

Ejemplo: habilitar o deshabilitar CloudWatch Contributor Insights for DynamoDB

La siguiente política de IAM concede permisos para habilitar o deshabilitar CloudWatch Contributor Insights for DynamoDB.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/contributorinsights.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBCloudWatchContributorInsights",
            "Condition": {"StringLike": {"iam:AWSServiceName":
 "contributorinsights.dynamodb.amazonaws.com"}}
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*"
```

```
        }
    ]
}
```

Para las tablas encriptadas por la clave KMS, el usuario necesita tener permisos kms:Decrypt para poder actualizar Contributor Insights.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-
service-role/contributorinsights.dynamodb.amazonaws.com/
AWSServiceRoleForDynamoDBCloudWatchContributorInsights",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName":
 "contributorinsights.dynamodb.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/*"
        },
        {
            "Effect": "Allow",
            "Resource": "arn:aws:kms:*:*:key/*",
            "Action": [
                "kms:Decrypt"
            ]
        }
    ]
}
```

Ejemplo: recuperar un informe de reglas de CloudWatch Contributor Insights

La siguiente política de IAM concede permisos para recuperar un informe de reglas de CloudWatch Contributor Insights.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetInsightRuleReport"
            ],
            "Resource": "arn:aws:cloudwatch:*:*:insight-rule/
DynamoDBContributorInsights*"
        }
    ]
}
```

Ejemplo: aplicar selectivamente permisos de CloudWatch Contributor Insights for DynamoDB basados en recursos

La siguiente política de IAM concede permisos para permitir las acciones `ListContributorInsights` y `DescribeContributorInsights` y deniega la acción `UpdateContributorInsights` para un índice secundario global concreto.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListContributorInsights",
                "dynamodb:DescribeContributorInsights"
            ],
```

```
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": [
                "dynamodb:UpdateContributorInsights"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/Books/
index/Author-index"
        }
    ]
}
```

Uso de roles vinculados a servicios para CloudWatch Contributor Insights for DynamoDB

CloudWatch Contributor Insights for DynamoDB utiliza roles vinculados a servicios de AWS Identity and Access Management (IAM). Un rol vinculado a un servicio es un tipo único de rol de IAM que está vinculado directamente a CloudWatch Contributor Insights for DynamoDB. Los roles vinculados a servicios están predefinidos por CloudWatch Contributor Insights for DynamoDB e incluyen todos los permisos que el servicio requiere para llamar a otros servicios de AWS en su nombre.

Un rol vinculado a un servicio simplifica la configuración de CloudWatch Contributor Insights for DynamoDB porque ya no tendrá que agregar manualmente los permisos necesarios. CloudWatch Contributor Insights for DynamoDB define los permisos de sus roles vinculados a servicios y, a menos que esté definido de otra manera, solo CloudWatch Contributor Insights for DynamoDB puede asumir sus roles. Los permisos definidos incluyen las políticas de confianza y de permisos, y que la política de permisos no se pueda asociar a ninguna otra entidad de IAM.

Para obtener información acerca de otros servicios que admiten roles vinculados a servicios, consulte Servicios de AWS que funcionan con IAM y busque los servicios que muestran Sí en la columna Rol vinculado a un servicio. Elija una opción Sí con un enlace para ver la documentación acerca del rol vinculado al servicio en cuestión.

Permisos del rol vinculado a servicios para CloudWatch Contributor Insights for DynamoDB

CloudWatch Contributor Insights for DynamoDB usa el rol vinculado al servicio denominado AWSServiceRoleForDynamoDBCloudWatchContributorInsights. El propósito del rol vinculado a servicios es permitir que Amazon DynamoDB administre las reglas de Amazon CloudWatch

Contributor Insights creadas para tablas e índices secundarios globales de DynamoDB en su nombre.

El rol vinculado al servicio `AWSServiceRoleForDynamoDBCloudWatchContributorInsights` depende de los siguientes servicios para asumir el rol:

- `contributorinsights.dynamodb.amazonaws.com`

La política de permisos del rol permite que CloudWatch Contributor Insights for DynamoDB realice las siguientes acciones en los recursos especificados:

- Acción: `Create and manage Insight Rules` en `DynamoDBContributorInsights`

Debe configurar permisos para permitir a una entidad de IAM (como un usuario, grupo o rol) crear, editar o eliminar un rol vinculado a servicios. Para obtener más información, consulte Permisos de roles vinculados a servicios en la Guía del usuario de IAM.

Creación de un rol vinculado a servicios para CloudWatch Contributor Insights for DynamoDB

No necesita crear manualmente un rol vinculado a servicios. Cuando habilita Contributor Insights en la Consola de administración de AWS, la AWS CLI o la API de AWS, CloudWatch Contributor Insights for DynamoDB le crea automáticamente un rol vinculado a un servicio.

Si elimina este rol vinculado a servicios y necesita crearlo de nuevo, puede utilizar el mismo proceso para volver a crear el rol en su cuenta. Cuando habilita Contributor Insights, CloudWatch Contributor Insights for DynamoDB le crea automáticamente un rol vinculado a un servicio de nuevo.

Edición de un rol vinculado a servicios para CloudWatch Contributor Insights for DynamoDB

CloudWatch Contributor Insights for DynamoDB no le permite editar el rol vinculado al servicio `AWSServiceRoleForDynamoDBCloudWatchContributorInsights`. Después de crear un rol vinculado al servicio, no podrá cambiar el nombre del rol, ya que varias entidades podrían hacer referencia al rol. Sin embargo, sí puede editar la descripción del rol con IAM. Para obtener más información, consulte Editar un rol vinculado a servicios en la Guía del usuario de IAM.

Eliminación de un rol vinculado a servicios para CloudWatch Contributor Insights for DynamoDB

No es necesario eliminar manualmente el rol de `AWSServiceRoleForDynamoDBCloudWatchContributorInsights`. Cuando desactiva

Contributor Insights en la Consola de administración de AWS, la AWS CLI o la API de AWS, CloudWatch Contributor Insights for DynamoDB limpia los recursos.

También puede utilizar la consola de IAM, la AWS CLI, la API de AWS para eliminar manualmente el rol vinculado al servicio. Para ello, primero debe limpiar manualmente los recursos del rol vinculado al servicio para poder eliminarlo después manualmente.

> **Note**
>
> Si el servicio de CloudWatch Contributor Insights for DynamoDB está utilizando el rol cuando intenta eliminar los recursos, la eliminación podría producir un error. En tal caso, espere unos minutos e intente de nuevo la operación.

Para eliminar manualmente el rol vinculado a servicios mediante IAM

Puede usar la consola de IAM, la AWS CLI o la API de AWS para eliminar el rol vinculado a un servicio de `AWSServiceRoleForDynamoDBCloudWatchContributorInsights`. Para obtener más información, consulte Eliminación de un rol vinculado a un servicio en la Guía del usuario de IAM.

Prácticas recomendadas para el diseño y la arquitectura con DynamoDB

Utilice esta sección para encontrar rápidamente recomendaciones que le permitan maximizar el rendimiento y minimizar los costos al usar DynamoDB.

Temas

- Diseño NoSQL para DynamoDB
- Uso del enfoque Well-Architected de DynamoDB para optimizar su carga de trabajo de DynamoDB
- Prácticas recomendadas para diseñar claves de partición y utilizarlas con eficacia en DynamoDB
- Prácticas recomendadas sobre el uso de claves de clasificación para organizar datos en DynamoDB
- Prácticas recomendadas para utilizar índices secundarios en DynamoDB
- Prácticas recomendadas para almacenar elementos y atributos grandes en DynamoDB
- Prácticas recomendadas para administrar los datos de serie temporal en DynamoDB
- Prácticas recomendadas para administrar las relaciones de varios a varios en las tablas de DynamoDB
- Prácticas recomendadas para consultar y analizar datos en DynamoDB
- Prácticas recomendadas para el diseño de tablas de DynamoDB
- Uso de tablas globales de DynamoDB
- Prácticas recomendadas para administrar el plano de control en DynamoDB
- Prácticas recomendadas para utilizar operaciones de datos masivos en DynamoDB
- Prácticas recomendadas para implementar el control de versiones en DynamoDB
- Prácticas recomendadas para interpretar los informes de facturación y uso de AWS en DynamoDB
- Migración de una tabla de DynamoDB de una cuenta a otra
- Recomendaciones para integrar DAX con las aplicaciones de DynamoDB

- Consideraciones sobre el uso de AWS PrivateLink para Amazon DynamoDB

Diseño NoSQL para DynamoDB

Los sistemas de bases de datos NoSQL como Amazon DynamoDB utilizan modelos alternativos de administración de datos; por ejemplo, pares clave-valor o almacenamiento de documentos. Al pasar de un sistema de administración de bases de datos relacionales a un sistema de bases de datos NoSQL como DynamoDB, es importante entender las principales diferencias y los enfoques de diseño específicos.

Temas

- Diferencias entre el diseño de datos relacionales y NoSQL
- Dos conceptos clave del diseño NoSQL
- Aproximación al diseño NoSQL
- NoSQL Workbench para DynamoDB

Diferencias entre el diseño de datos relacionales y NoSQL

Los sistemas de bases de datos relacionales (RDBMS) y las bases de datos NoSQL tienen diferentes ventajas y desventajas:

- En RDBMS, los datos se pueden consultar de manera flexible, pero las consultas son relativamente costosas y no se escalan bien cuando hay mucho tráfico (consulte Primeros pasos para modelar datos relacionales en DynamoDB).
- En una base de datos NoSQL como DynamoDB, existe un número limitado de métodos para consultar los datos; si se utiliza cualquier otro método diferente a estos, resultará más costoso y lento realizar las consultas.

Estas diferencias hacen que el diseño de las bases de datos sea muy distinto entre los dos sistemas:

- En RDBMS, el diseño busca la flexibilidad sin preocuparse por los detalles o el rendimiento de la implementación. Por lo general, la optimización de consultas no afecta al diseño del esquema, pero la normalización es importante.
- En DynamoDB, el esquema se diseña específicamente para que las consultas más habituales y más importantes resulten lo más rápidas y económicas posible. Las estructuras de datos se ajustan a los requisitos específicos de los casos de uso de la organización.

Dos conceptos clave del diseño NoSQL

El diseño NoSQL requiere un modo de pensar distinto al diseño de RDBMS. En un sistema RDBMS, puede empezar a crear un modelo de datos normalizados sin pensar en los patrones de acceso. Posteriormente, podrá ampliar este modelo cuando surjan nuevos requisitos sobre preguntas y consultas. Puede organizar cada tipo de datos en su propia tabla.

Diferencias del diseño NoSQL

- Por el contrario, no debería comenzar a diseñar su esquema para DynamoDB hasta que sepa las preguntas a las que tiene que responder. Es esencial conocer de antemano los problemas del negocio y los casos de uso de la aplicación.
- Debe mantener el menor número de tablas posible en una aplicación de DynamoDB. Tener menos tablas hace que las cosas sean más escalables, requiere menos administración de permisos y reduce la sobrecarga de la aplicación de DynamoDB. También puede ayudar a mantener los costos de las copias de seguridad generalmente más bajos.

Aproximación al diseño NoSQL

El primer paso para diseñar la aplicación de DynamoDB es identificar los patrones de consulta específicos que el sistema debe satisfacer.

En particular, antes de empezar, es importante entender tres propiedades fundamentales de los patrones de acceso de la aplicación:

- Tamaño de los datos: saber cuántos datos se almacenarán y solicitarán a la vez ayudará a determinar el método más eficaz para particionarlos.
- Forma de los datos: en lugar de dar forma a los datos al procesar las consultas (como ocurre en los sistemas RDBMS), las bases de datos NoSQL organizan los datos de modo que la forma que tienen en la base de datos se corresponda con la que se va a consultar. Este es un factor crucial para aumentar la velocidad y la escalabilidad.
- Velocidad de los datos: DynamoDB escala aumentando el número de particiones físicas disponibles para procesar las consultas y distribuyendo eficazmente los datos entre esas particiones. Conocer de antemano cuáles serán las cargas de consulta máximas puede ayudar a determinar cómo deben particionarse los datos para hacer un uso óptimo de la capacidad de E/S.

Después de identificar los requisitos de consulta específicos, puede organizar los datos con arreglo a los principios generales que rigen el rendimiento:

- Agrupar los datos relacionales. Las investigaciones han demostrado que el principio de "localidad de referencia", que mantiene los datos relacionados juntos en un solo lugar, es un factor clave para mejorar el rendimiento y los tiempos de respuesta en los sistemas NoSQL, del mismo modo que se descubrió que era importante para optimizar las tablas de enrutamiento hace muchos años.

  Como regla general, en una aplicación de DynamoDB, debe mantenerse el menor número de tablas posible.

  Las excepciones son aquellos casos en los que hay implicados datos de serie temporal de gran volumen o conjuntos de datos que tienen patrones de acceso muy diferentes. Normalmente, basta una sola tabla con índices invertidos para permitir que, a través de consultas simples, se creen y recuperen las estructuras de datos jerárquicas y complejas que necesita la aplicación.
- Utilizar un orden de clasificación. Los elementos relacionados pueden agruparse y consultarse de forma eficaz si su diseño de claves hace que se ordenen juntos. Esta es una estrategia importante en el diseño NoSQL.
- Distribuir consultas. También es importante que no haya una gran cantidad de consultas concentradas en la misma parte de la base de datos, donde podría sobrepasarse la capacidad de E/S. En su lugar, debe diseñar claves de datos que distribuyan el tráfico lo más uniformemente posible entre las particiones y eviten la creación de puntos calientes.
- Utilizar índices secundarios globales. Si crea índices secundarios globales específicos, puede permitir el uso de diferentes consultas en la tabla principal, lo que sigue siendo rápido y relativamente económico.

Estos principios generales se traducen en unos patrones de diseño comunes que puede utilizar para modelar los datos de forma eficaz en DynamoDB.

NoSQL Workbench para DynamoDB

NoSQL Workbench para DynamoDB es una aplicación GUI de cliente multiplataforma que puede usar para el desarrollo moderno de bases de datos y operaciones. Está disponible para Windows, macOS y Linux. NoSQL Workbench es una herramienta de desarrollo visual que proporciona características de modelado de datos, visualización de datos, generación de datos de muestra y desarrollo de consultas para ayudarle a diseñar, crear, consultar y administrar tablas de DynamoDB. Con NoSQL Workbench para DynamoDB, puede crear nuevos modelos de datos o diseñar modelos

basados en modelos de datos existentes que satisfagan los patrones de acceso a datos de su aplicación. También puede importar y exportar el modelo de datos diseñado al final del proceso. Para obtener más información, consulte Creación de modelos de datos con NoSQL Workbench.

Uso del enfoque Well-Architected de DynamoDB para optimizar su carga de trabajo de DynamoDB

En esta sección se describe el enfoque Well-Architected de Amazon DynamoDB, una colección de principios de diseño y directrices para diseñar cargas de trabajo de DynamoDB bien diseñadas.

Optimización de los costos en las tablas de DynamoDB

En esta sección se describen las prácticas recomendadas para optimizar los costos de sus tablas de DynamoDB existentes. Debería examinar las siguientes estrategias para ver qué estrategia de optimización de costos se adapta mejor a sus necesidades y abordarlas de forma iterativa. Cada estrategia proporcionará una visión general de lo que puede estar afectando a sus costos, qué señales buscar y una orientación prescriptiva sobre cómo reducirlos.

Temas

- Evaluación de los costos en el nivel de tabla
- Evaluación del modo de capacidad de una tabla de DynamoDB
- Evaluación de la configuración de escalado automático de la tabla de DynamoDB
- Evaluación de la selección de clases de tabla de DynamoDB
- Identificación de recursos no utilizados en DynamoDB
- Evaluación de los patrones de uso de las tablas de DynamoDB
- Evaluación del uso de los flujos de DynamoDB
- Evaluación de la capacidad aprovisionada para lograr un aprovisionamiento del tamaño adecuado en la tabla de DynamoDB

Evaluación de los costos en el nivel de tabla

La herramienta Explorador de costos que se encuentra en la Consola de administración de AWS le permite ver los costos desglosados por tipo, como los gastos de lectura, escritura, almacenamiento y copia de seguridad. También puede ver estos costos resumidos por periodo, por ejemplo, por mes o por día.

Un reto al que pueden enfrentarse los administradores es cuando hay que revisar los costos de solo una tabla concreta. Algunos de estos datos están disponibles mediante la consola de DynamoDB o a través de llamadas a la API `DescribeTable`; no obstante, el Explorador de costos no permite, de forma predeterminada, filtrar ni agrupar por costos asociados a una tabla específica. En esta sección se mostrará cómo utilizar el etiquetado para realizar un análisis de costos de tablas individuales en el Explorador de costos.

Temas

- Cómo ver los costos de una sola tabla de DynamoDB
- Vista predeterminada del Explorador de costos
- Cómo utilizar y aplicar las etiquetas de tabla en el Explorador de costos

Cómo ver los costos de una sola tabla de DynamoDB

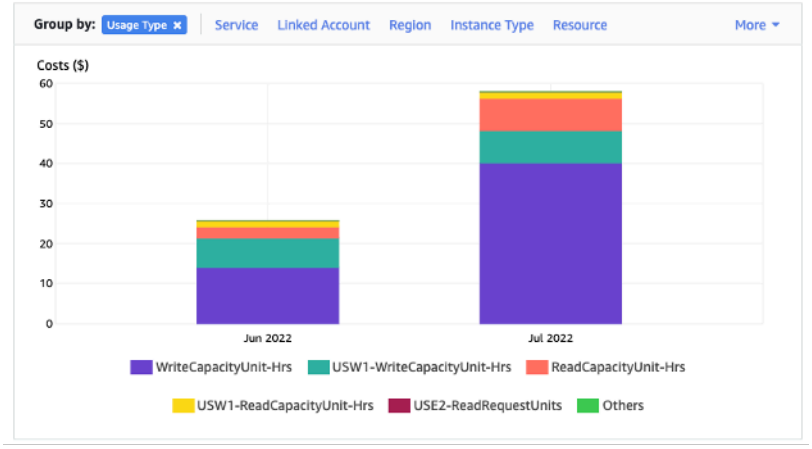
Tanto la Consola de administración de AWS de Amazon DynamoDB como la API `DescribeTable` le mostrarán información sobre una sola tabla, incluido el esquema de la clave principal, cualquier índice de la tabla y el tamaño y el recuento de elementos de la tabla y de cualquier índice. El tamaño de la tabla, más el tamaño de los índices, puede utilizarse para calcular el costo mensual de almacenamiento de su tabla. Por ejemplo, 0,25 USD por GB en la región us-east-1.

Si la tabla se encuentra en modo de capacidad aprovisionada, se devuelven también las configuraciones actuales de RCU y WCU. Se podrían utilizar para calcular los costos actuales de lectura y escritura de la tabla, pero estos costos podrían cambiar, sobre todo si la tabla se ha configurado con el escalado automático.

> **Note**
>
> Si la tabla se encuentra en el modo de capacidad bajo demanda, `DescribeTable` no ayudará a estimar los costos de rendimiento, ya que estos se facturan por el uso real, no aprovisionado, en un periodo determinado.

Vista predeterminada del Explorador de costos

La vista predeterminada del Explorador de costos ofrece gráficos que muestran el costo de los recursos consumidos, como el rendimiento y el almacenamiento. Puede elegir agrupar los costos por periodo, como los totales por mes o por día. También se pueden desglosar y comparar los costos de almacenamiento, lecturas, escrituras y otras características.



Cómo utilizar y aplicar las etiquetas de tabla en el Explorador de costos

De forma predeterminada, el Explorador de costos no proporciona un resumen de los costos de una tabla específica, ya que combinará los costos de varias tablas en un total. No obstante, puede utilizar el etiquetado de recursos de AWS para identificar cada tabla con una etiqueta de metadatos. Las etiquetas son pares de clave-valor que puede utilizar para diversos fines, como identificar todos los recursos que pertenecen a un proyecto o departamento. En este ejemplo, supondremos que tiene una tabla llamada MyTable.

1. Establezca una etiqueta con la clave de table_name y el valor de MyTable.
2. Active la etiqueta en el Explorador de costos y, a continuación, filtre el valor de la etiqueta para obtener una mayor visibilidad de los costos de cada tabla.

> **Note**
>
> La etiqueta puede tardar uno o dos días en comenzar a aparecer en el Explorador de costos

Puede establecer las etiquetas de metadatos por su cuenta en la consola o a través de automatización como la CLI de AWS o el SDK de AWS. Considere la posibilidad de exigir que se establezca una etiqueta table_name como parte del proceso de creación de una nueva tabla en su organización. En el caso de las tablas existentes, existe una utilidad de Python que encontrará y aplicará estas etiquetas a todas las tablas existentes en una región determinada de su cuenta. Consulte Eponymous Table Tagger en GitHub para obtener más detalles.

Evaluación del modo de capacidad de una tabla de DynamoDB

En esta sección se ofrece una visión general de cómo seleccionar el modo de capacidad adecuado para su tabla de DynamoDB. Cada modo está ajustado para satisfacer las necesidades de una carga de trabajo diferente en cuanto a la capacidad de respuesta a los cambios en el rendimiento, así como a la forma de facturar ese uso. Debe sopesar estos factores al tomar su decisión.

Temas

- Qué modos de capacidad de tabla hay disponibles
- Cuándo seleccionar el modo de capacidad bajo demanda
- Cuándo seleccionar el modo de capacidad aprovisionada
- Factores adicionales que se deben tener en cuenta al elegir un modo de capacidad de tabla

Qué modos de capacidad de tabla hay disponibles

Cuando cree una tabla de DynamoDB, debe seleccionar el modo de capacidad bajo demanda o aprovisionada.

Puede cambiar las tablas del modo de capacidad aprovisionada al modo bajo demanda hasta cuatro veces en un periodo continuo de 24 horas. Las tablas pueden cambiar del modo bajo demanda al modo de capacidad aprovisionada en cualquier momento.

Modo de capacidad bajo demanda

El modo de capacidad bajo demanda se ha diseñado para eliminar la necesidad de planificar o aprovisionar la capacidad de la tabla de DynamoDB. En este modo, su tabla adaptará instantáneamente las solicitudes a su tabla sin necesidad de escalar verticalmente ningún recurso (hasta el doble del rendimiento máximo anterior de la tabla).

DynamoDB bajo demanda ofrece precios de pago por solicitud para las solicitudes de lectura y escritura. De este modo, únicamente tendrá que pagar por aquello que utilice.

Modo de capacidad aprovisionada

El modo de capacidad aprovisionada es un modelo más tradicional en el que se debe definir cuánta capacidad tiene disponible la tabla para las solicitudes, ya sea directamente o con la ayuda del escalado automático. Dado que se aprovisiona una capacidad específica para la tabla en un momento dado, la facturación se basa en la capacidad aprovisionada total y no en el número de

solicitudes consumidas. Superar la capacidad asignada también puede provocar que la tabla rechace solicitudes y reduzca la experiencia de los usuarios de sus aplicaciones.

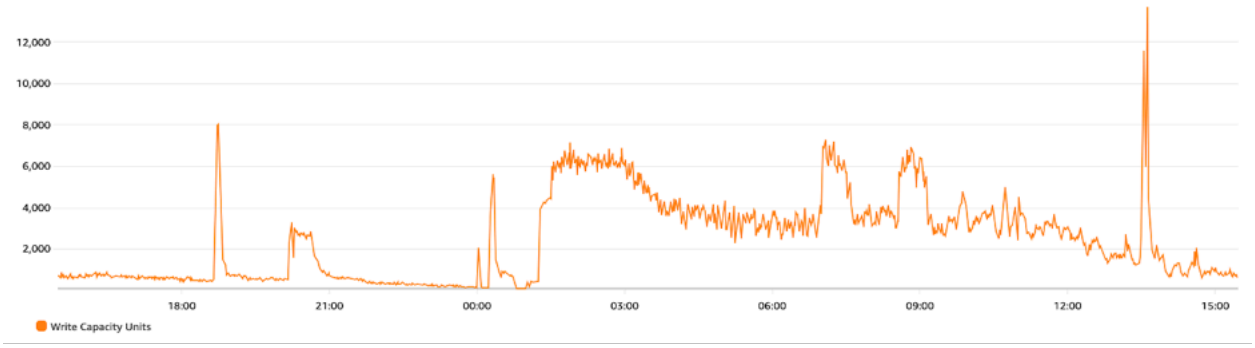
El modo de capacidad aprovisionada requiere una supervisión constante para encontrar un equilibrio entre no aprovisionar en exceso ni en defecto la tabla para mantener tanto la limitación baja como los costos ajustados.

Cuándo seleccionar el modo de capacidad bajo demanda

A la hora de optimizar el costo, el modo bajo demanda es la mejor opción cuando se tiene una carga de trabajo similar a la de los siguientes gráficos.

Los siguientes factores contribuyen a este tipo de carga de trabajo:

- Patrón de tráfico que evoluciona con el tiempo
- Volumen variable de solicitudes (resultante de las cargas de trabajo por lotes)
- Tiempo de solicitud imprevisible (lo que provoca picos de tráfico)
- Cae a cero o por debajo del 30 % del pico para una hora determinada



En el caso de las cargas de trabajo con los factores mencionados, si se utiliza el escalado automático para mantener una capacidad suficiente en la tabla para responder a los picos de tráfico, es probable que la tabla esté aprovisionada en exceso y cueste más de lo necesario o que la tabla esté poco aprovisionada y las solicitudes se limiten innecesariamente. El modo de capacidad bajo demanda es la mejor opción porque puede gestionar el tráfico fluctuante sin necesidad de predecir ni ajustar la capacidad.

Con el modelo bajo demanda de precios de pago por solicitud, no tiene que preocuparse por la capacidad inactiva porque solo paga por el rendimiento que realmente utiliza. Se le factura por solicitud de lectura o escritura consumida, por lo que los costos reflejan directamente el uso real, lo que facilita el equilibrio entre costos y rendimiento. Si lo desea, también puede configurar el

rendimiento máximo de lectura o escritura (o ambos) por segundo para tablas individuales bajo demanda e índices secundarios globales para ayudar a mantener limitados los costos y el uso. Para obtener más información, consulte Rendimiento máximo de las tablas bajo demanda.

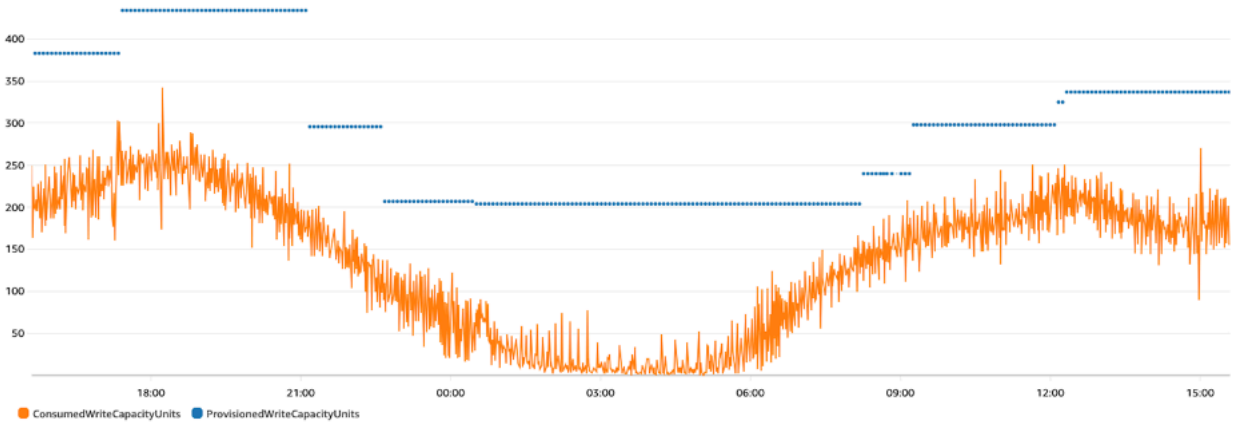
Cuándo seleccionar el modo de capacidad aprovisionada

Una carga de trabajo ideal para el modo de capacidad aprovisionada es aquella con un patrón de uso más estable y predecible como el del gráfico siguiente.

> **Note**
>
> Recomendamos revisar las métricas en un período detallado, por ejemplo, en 14 días, antes de tomar medidas con respecto a la capacidad aprovisionada.

Los siguientes factores contribuyen a este tipo de carga de trabajo:

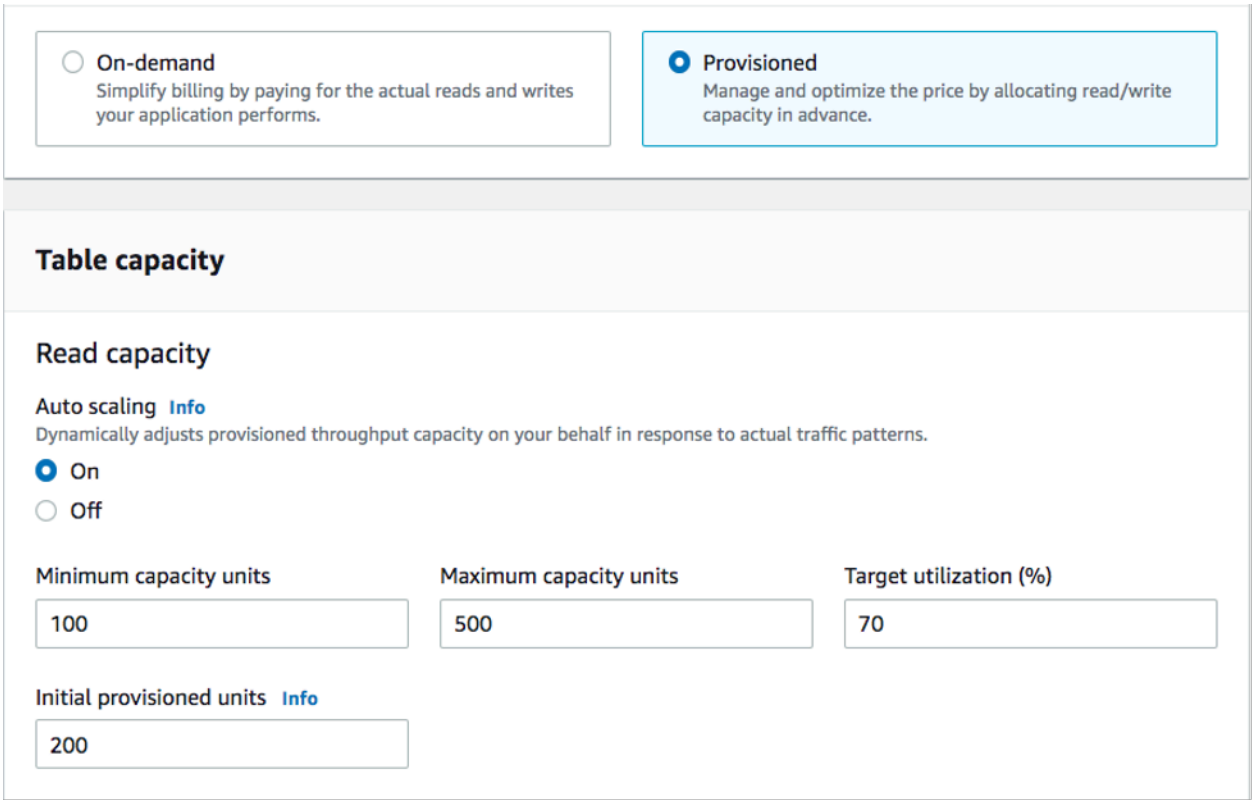
- Tráfico constante, previsible y cíclico durante una hora o un día determinados
- Ampliaciones limitadas de tráfico a corto plazo



Como los volúmenes de tráfico en una hora o un día determinados son más estables, puede establecer la capacidad aprovisionada de la tabla relativamente cerca de la capacidad real consumida de la tabla. La optimización de costos de una tabla de capacidad aprovisionada es, en última instancia, un ejercicio para conseguir que la capacidad aprovisionada (línea azul) se acerque lo máximo posible a la capacidad consumida (línea naranja) sin aumentar `ThrottledRequests` en la tabla. El espacio entre las dos líneas es tanto una capacidad desperdiciada como un seguro contra una experiencia de usuario deficiente debido a la limitación. Si puede predecir los requisitos de

rendimiento de la aplicación y prefiere la previsibilidad de costos que supone controlar la capacidad de lectura y escritura, quizá le convenga seguir utilizando tablas aprovisionadas.

DynamoDB proporciona un escalado automático para las tablas de capacidad aprovisionada que realizará el equilibrio automáticamente en su nombre. Esto le permite hacer un seguimiento de su capacidad consumida a lo largo del día y establecer la capacidad de la tabla en función de unas pocas variables. Si utiliza el escalado automático, la tabla estará sobreaprovisionada y tendrá que afinar con precisión la relación entre el número de limitaciones y las unidades de capacidad sobreaprovisionadas para que coincida con las necesidades de la carga de trabajo.



Unidades de capacidad mínima

Puede establecer la capacidad mínima de una tabla para disminuir la limitación, pero no reducirá el costo de la tabla. Si su tabla tiene periodos de baja utilización seguidos de una repentina ampliación de la utilización, la configuración del mínimo puede evitar que el escalado automático establezca una capacidad de la tabla demasiado baja.

Unidades de capacidad máxima

Puede establecer la capacidad máxima de una tabla para limitar el escalado de una tabla por encima de lo previsto. Considere la posibilidad de aplicar un máximo para las tablas de desarrollo o de prueba cuando no se desee realizar pruebas de carga a gran escala. Puede establecer un máximo

para cualquier tabla, pero asegúrese de evaluar periódicamente esta configuración con respecto a la línea de base de la tabla cuando la utilice en producción para evitar una limitación accidental.

Objetivo de utilización

El establecimiento de la utilización objetivo de la tabla es el principal medio de optimización de costos para una tabla de capacidad aprovisionada. Si establece aquí un valor de porcentaje más bajo, aumentará el exceso de aprovisionamiento de la tabla, con lo que se incrementará el costo, pero se reducirá el riesgo de limitación. Si establece un valor de porcentaje más alto, disminuirá el exceso de aprovisionamiento de la tabla, pero aumentará el riesgo de limitación.

Factores adicionales que se deben tener en cuenta al elegir un modo de capacidad de tabla

Al decidir entre los dos modos, hay algunos factores adicionales que merece la pena tener en cuenta.

Utilización de la capacidad aprovisionada

Para determinar cuándo el modo bajo demanda costaría menos que la capacidad aprovisionada, es útil observar la utilización de la capacidad aprovisionada, que se refiere a la eficacia con la que se utilizan los recursos asignados (o “aprovisionados”). El modo bajo demanda cuesta menos para cargas de trabajo con una utilización media de la capacidad aprovisionada inferior al 35 %. En muchos casos, incluso para cargas de trabajo con una utilización de la capacidad aprovisionada superior al 35 %, puede resultar más rentable utilizar el modo bajo demanda, especialmente si la carga de trabajo tiene periodos de baja actividad mezclados con picos ocasionales.

Capacidad reservada

En el caso de las tablas de capacidad aprovisionada, DynamoDB ofrece la posibilidad de comprar capacidad reservada para su capacidad de lectura y escritura (las unidades de capacidad de escritura replicada [rWCU] y las tablas Estándar - Acceso poco frecuente no cumplen los requisitos actualmente). La capacidad reservada ofrece importantes descuentos sobre los precios de la capacidad aprovisionada estándar.

A la hora de decidir entre los dos modos de tabla, plantéese en qué medida este descuento adicional afectará al costo de la tabla. En algunos casos, puede costar menos ejecutar una carga de trabajo relativamente impredecible puede ser más barato ejecutarla en una tabla de capacidad aprovisionada con capacidad reservada.

Mejorar la previsibilidad de la carga de trabajo

En algunas situaciones, una carga de trabajo puede tener aparentemente un patrón tanto predecible como impredecible. Aunque esto se puede apoyar fácilmente con una tabla bajo demanda, es probable que los costos sean mejores si se pueden mejorar los patrones impredecibles de la carga de trabajo.

Una de las causas más comunes de estos patrones son las importaciones por lotes. Este tipo de tráfico puede superar a menudo la capacidad de línea de base de la tabla, hasta tal punto que se produciría una limitación si se ejecutara. Para mantener una carga de trabajo como esta en una tabla de capacidad aprovisionada, considere las siguientes opciones:

- Si el lote se produce en horas programadas, puede programar un aumento de la capacidad de escalado automático antes de que se ejecute
- Si el lote se produce de forma aleatoria, considere la posibilidad de intentar prolongar el tiempo de ejecución en lugar de ejecutarlo lo más rápido posible
- Agregue un periodo de aceleración a la importación en el que la velocidad de dicha importación comience siendo pequeña pero se incremente lentamente durante unos minutos hasta que el escalado automático haya tenido la oportunidad de comenzar a ajustar la capacidad de la tabla

Evaluación de la configuración de escalado automático de la tabla de DynamoDB

En esta sección se proporciona información general de cómo evaluar la configuración de escalado automático en las tablas de DynamoDB. El escalado automático de Amazon DynamoDB es una función que administra el rendimiento de las tablas y del índice secundario global (GSI) en función del tráfico de la aplicación y de la métrica de utilización objetivo. Esto garantiza que las tablas o GSI tengan la capacidad necesaria para los modelos de la aplicación.

El servicio de AWS Auto Scaling monitoreará el uso actual de la tabla y lo comparará con el valor de uso objetivo: `TargetValue`. Le notificará si es el momento de aumentar o disminuir la capacidad asignada.

Temas

- Entender la configuración de escalado automático
- Cómo identificar tablas con una utilización objetivo baja (<=50 %)
- Cómo abordar las cargas de trabajo con variaciones estacionales
- Cómo abordar cargas de trabajo con picos con patrones desconocidos
- Cómo abordar las cargas de trabajo con aplicaciones enlazadas

Entender la configuración de escalado automático

Definir el valor correcto para el uso objetivo, el paso inicial y los valores finales es una actividad que requiere la participación del equipo de operaciones. Esto le permite definir correctamente los valores en función del uso histórico de la aplicación, que se utilizará para desencadenar las políticas de AWS Auto Scaling. El objetivo de uso es el porcentaje de la capacidad total que se debe alcanzar durante un periodo de tiempo antes de que se apliquen las reglas de escalado automático.

Cuando se establece un objetivo de uso alto (un objetivo alrededor del 90 %), significa que el tráfico debe superar el 90 % durante un periodo de tiempo antes de que se active el escalado automático. No debe usar un objetivo de alta utilización a menos que la aplicación sea muy constante y no reciba picos de tráfico.

Cuando se establece una utilización muy baja (un objetivo inferior al 50 %), significa que la aplicación tendría que alcanzar el 50 % de la capacidad aprovisionada antes de desencadenar una política de escalado automático. A menos que el tráfico de la aplicación crezca a un ritmo muy agresivo, esto normalmente se traduce en capacidad no utilizada y en recursos desperdiciados.

Cómo identificar tablas con una utilización objetivo baja (<=50 %)

Puede utilizar AWS CLI o Consola de administración de AWS para monitorear e identificar `TargetValues` para las políticas de escalado automático en los recursos de DynamoDB:

AWS CLI

1. Devuelva la lista completa de recursos mediante la ejecución del siguiente comando:

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    dynamodb
   ```

   Este comando devolverá la lista completa de políticas de escalado automático que se emiten en cualquier recurso de DynamoDB. Si solo desea recuperar los recursos de una tabla en particular, puede agregar `–resource-id parameter`. Por ejemplo:

   ```
   aws application-autoscaling describe-scaling-policies --service-namespace
    dynamodb --resource-id "table/<table-name>”
   ```

2. Devolver solo las políticas de escalado automático para un GSI en particular mediante la ejecución del siguiente comando

```
aws application-autoscaling describe-scaling-policies --service-namespace
 dynamodb --resource-id "table/<table-name>/index/<gsi-name>”
```

Los valores que nos interesan para las políticas de escalado automático se destacan a continuación. Queremos asegurarnos de que el valor objetivo sea superior al 50 % para evitar un aprovisionamiento excesivo. Debería obtener un resultado similar al siguiente:

```
{
    "ScalingPolicies": [
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/dynamodb/table/<table-name>/index/<index-
name>:policyName/$<full-gsi-name>-scaling-policy",
            "PolicyName": $<full-gsi-name>”,
            "ServiceNamespace": "dynamodb",
            "ResourceId": "table/<table-name>/index/<index-name>",
            "ScalableDimension": "dynamodb:index:WriteCapacityUnits",
            "PolicyType": "TargetTrackingScaling",
            "TargetTrackingScalingPolicyConfiguration": {
                "TargetValue": 70.0,
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "DynamoDBWriteCapacityUtilization"
                }
            },
            "Alarms": [
                ...
            ],
            "CreationTime": "2022-03-04T16:23:48.641000+10:00"
        },
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/dynamodb/table/<table-name>/index/<index-
name>:policyName/$<full-gsi-name>-scaling-policy",
            "PolicyName":$<full-gsi-name>”,
            "ServiceNamespace": "dynamodb",
            "ResourceId": "table/<table-name>/index/<index-name>",
            "ScalableDimension": "dynamodb:index:ReadCapacityUnits",
            "PolicyType": "TargetTrackingScaling",
            "TargetTrackingScalingPolicyConfiguration": {
                "TargetValue": 70.0,
                "PredefinedMetricSpecification": {
```

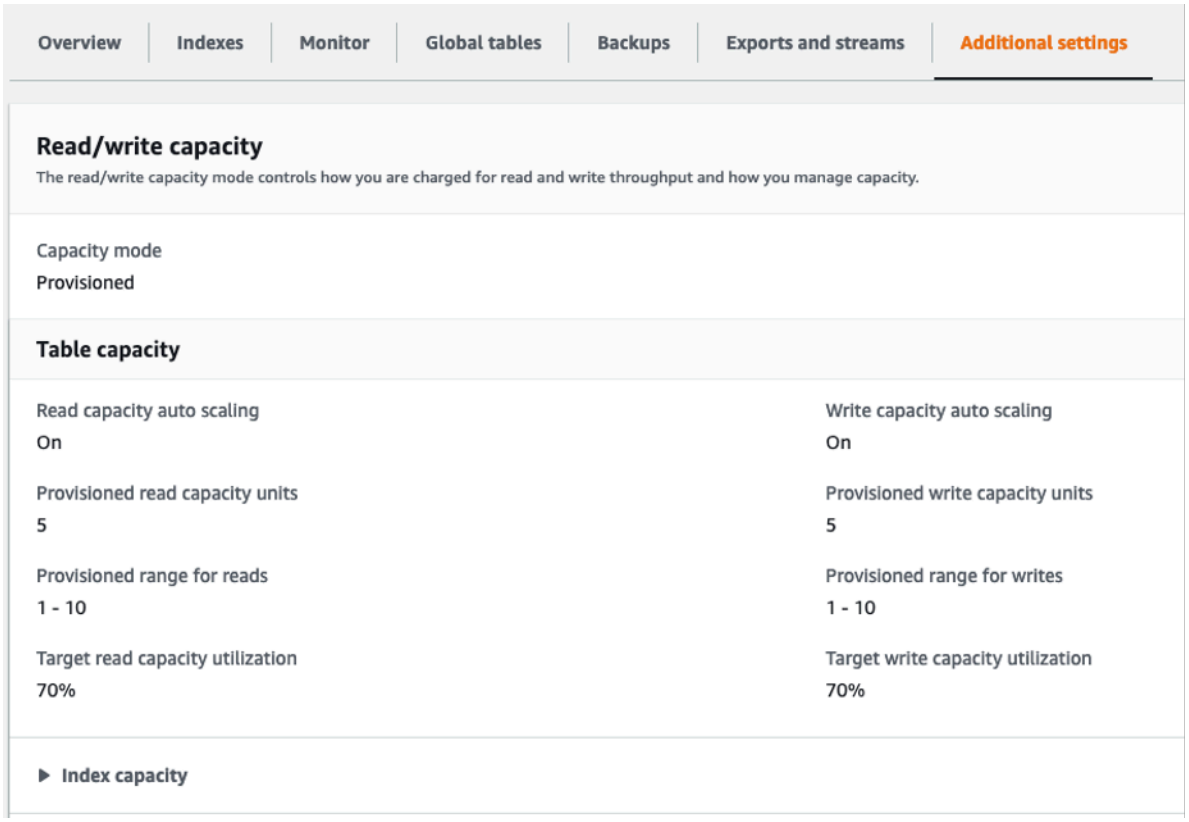
```
                    "PredefinedMetricType": "DynamoDBReadCapacityUtilization"
                }
            },
            "Alarms": [
                ...
            ],
            "CreationTime": "2022-03-04T16:23:47.820000+10:00"
        }
    ]
}
```

Consola de administración de AWS

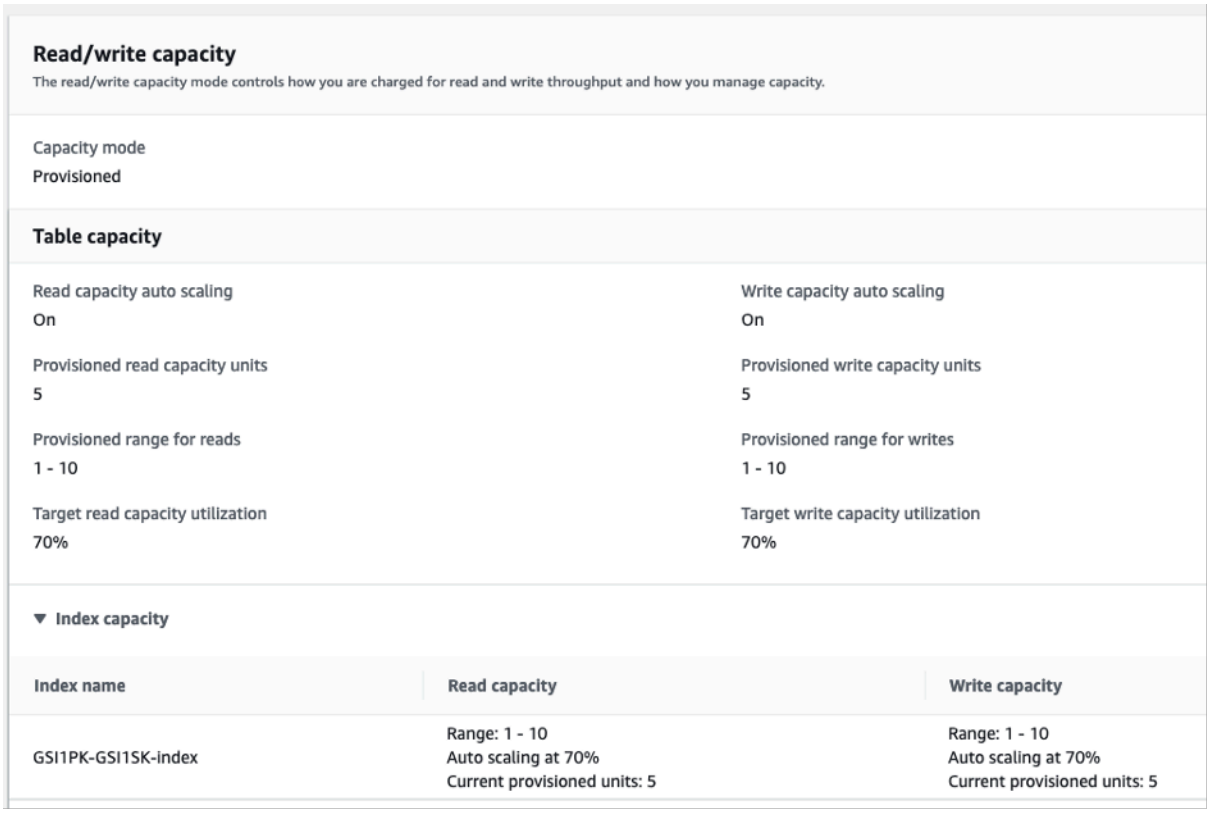
1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.

   Seleccione la Región de AWS apropiada si es necesario.

2. En la barra de navegación izquierda, seleccione Tablas. En la página Tablas, seleccione el Nombre de la tabla.
3. En la página Detalles de la tabla, elija Configuración adicional y luego revise la configuración de escalado automático de la tabla.



Para los índices, expanda la sección Capacidad de índice para revisar la configuración de escalado automático del índice.



Si los valores de utilización objetivo son inferiores o iguales al 50 %, debe analizar las métricas de uso de la tabla para ver si están subaprovisionadas o sobreaprovisionadas.

Cómo abordar las cargas de trabajo con variaciones estacionales

Considere el siguiente escenario: la aplicación funciona por debajo de un valor medio mínimo la mayor parte del tiempo, pero el objetivo de uso es bajo, por lo que la aplicación puede reaccionar rápidamente ante los eventos que se producen a determinadas horas del día y usted tiene suficiente capacidad y evita que se limite. Este escenario es habitual cuando una aplicación está muy ocupada durante el horario de oficina normal (de 9:00 a 17:00), pero que luego funciona a un nivel básico fuera del horario laboral. Dado que algunos usuarios empezarán a conectarse antes de las 9:00, la aplicación utiliza este umbral bajo para aumentar rápidamente y alcanzar la capacidad requerida durante las horas pico.

Este escenario podría ser así:

- Entre las 17:00 y las 9:00, las unidades de `ConsumedWriteCapacity` permanecen entre 90 y 100

- Los usuarios comienzan a conectarse a la aplicación antes de las 9:00 y la capacidad de las unidades aumenta considerablemente (el valor máximo que ha visto es de 1500 WCU)
- Por término medio, el uso de la aplicación varía entre 800 y 1200 durante las horas de trabajo

Si se encuentra con el escenario anterior, considere la posibilidad de utilizar el escalado automático programado, en el que la tabla aún podría tener configurada una regla de escalado automático de aplicaciones, pero con una utilización objetivo menos agresiva que solo aprovisione la capacidad adicional en los intervalos específicos que necesite.

Puede usar la AWS CLI para ejecutar los siguientes pasos para crear una regla de escalado automático programada que se ejecutará en función de la hora del día y el día de la semana.

1. Registre la tabla de DynamoDB o GSI como objetivo escalable con Application Auto Scaling. Un destino escalable es un recurso que Application Auto Scaling puede escalar horizontalmente o escalar verticalmente.

   ```
   aws application-autoscaling register-scalable-target \
       --service-namespace dynamodb \
       --scalable-dimension dynamodb:table:WriteCapacityUnits \
       --resource-id table/<table-name> \
       --min-capacity 90 \
       --max-capacity 1500
   ```

2. Configure acciones programadas según los requisitos.

   Necesitaremos dos reglas para cubrir el escenario: una para escalar verticalmente y otra para reducir verticalmente. La primera regla para escalar verticalmente la acción programada:

   ```
   aws application-autoscaling put-scheduled-action \
       --service-namespace dynamodb \
       --scalable-dimension dynamodb:table:WriteCapacityUnits \
       --resource-id table/<table-name> \
       --scheduled-action-name my-8-5-scheduled-action \
       --scalable-target-action MinCapacity=800,MaxCapacity=1500 \
       --schedule "cron(45 8 ? * MON-FRI *)" \
       --timezone "Australia/Brisbane"
   ```

   La segunda regla para reducir verticalmente la acción programada:

   ```
   aws application-autoscaling put-scheduled-action \
   ```

```
    --service-namespace dynamodb \
    --scalable-dimension dynamodb:table:WriteCapacityUnits \
    --resource-id table/<table-name> \
    --scheduled-action-name my-5-8-scheduled-down-action \
    --scalable-target-action MinCapacity=90,MaxCapacity=1500 \
    --schedule "cron(15 17 ? * MON-FRI *)" \
    --timezone "Australia/Brisbane"
```

3. Ejecute el siguiente comando para validar que ambas reglas se han activado:

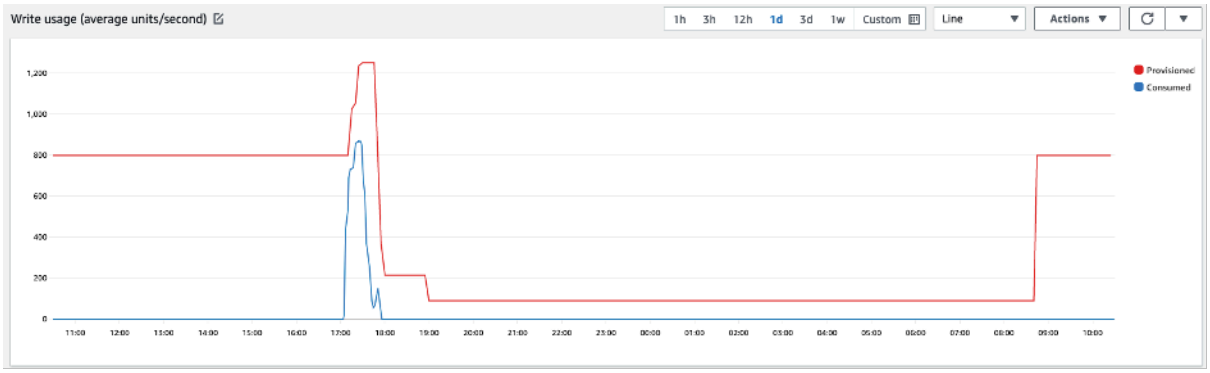
```
aws application-autoscaling describe-scheduled-actions --service-namespace dynamodb
```

Debería obtener un resultado como este:

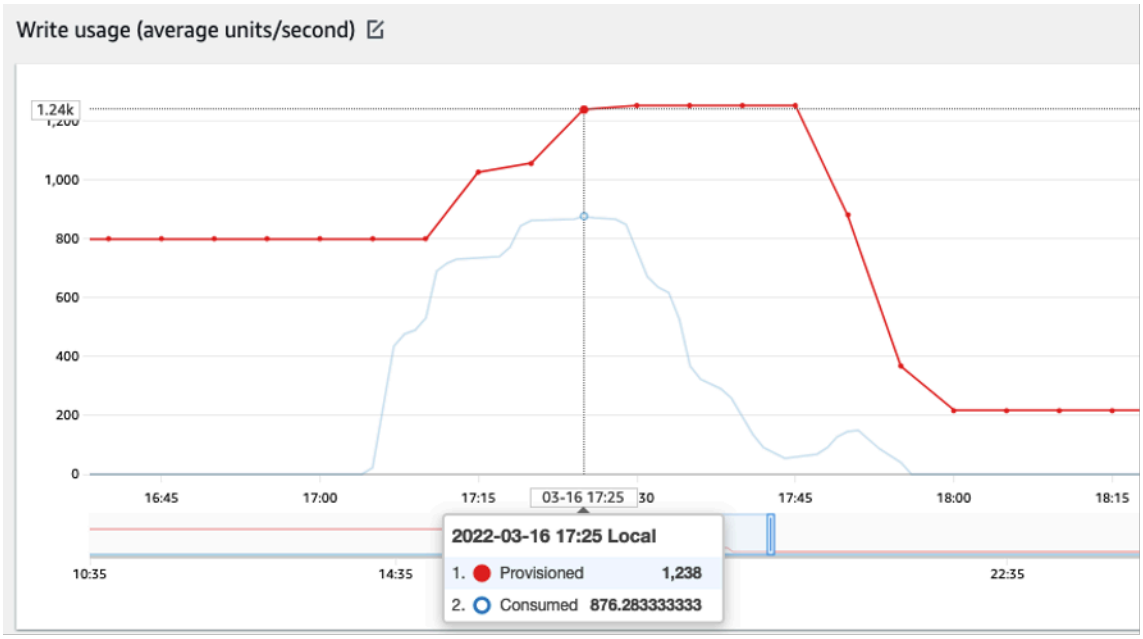
```
{
    "ScheduledActions": [
        {
            "ScheduledActionName": "my-5-8-scheduled-down-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
table/<table-name>:scheduledActionName/my-5-8-scheduled-down-action",
            "ServiceNamespace": "dynamodb",
            "Schedule": "cron(15 17 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
            "ResourceId": "table/<table-name>",
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "ScalableTargetAction": {
                "MinCapacity": 90,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:30:25.100000+10:00"
        },
        {
            "ScheduledActionName": "my-8-5-scheduled-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/dynamodb/
table/<table-name>:scheduledActionName/my-8-5-scheduled-action",
            "ServiceNamespace": "dynamodb",
            "Schedule": "cron(45 8 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
            "ResourceId": "table/<table-name>",
            "ScalableDimension": "dynamodb:table:WriteCapacityUnits",
            "ScalableTargetAction": {
```

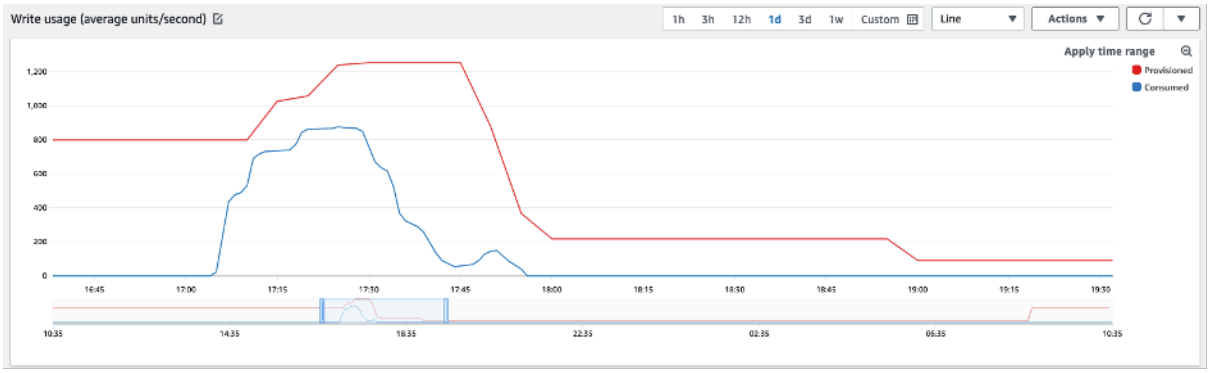
```
                "MinCapacity": 800,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:28:57.816000+10:00"
        }
    ]
}
```

La siguiente imagen muestra un ejemplo de carga de trabajo que siempre mantiene el objetivo de utilización del 70 %. Observe cómo las reglas de escalado automático se siguen aplicando y el rendimiento no se reducirá.



Al ampliar el zoom, podemos ver que ha habido un aumento en la aplicación que ha desencadenado el umbral de escalado automático del 70 %, lo que obligó a que el escalado automático entrara en vigor y proporcionara la capacidad adicional necesaria para la tabla. La acción de escalado automático programada afectará a los valores máximo y mínimo y es su responsabilidad configurarlos.





Cómo abordar cargas de trabajo con picos con patrones desconocidos

En este escenario, la aplicación utiliza un objetivo de uso muy bajo porque aún no conoce los patrones de la aplicación y quiere asegurarse de que la carga de trabajo no esté limitada.

Considere utilizar el modo de capacidad bajo demanda en su lugar. Las tablas bajo demanda son perfectas para cargas de trabajo con picos en los que no se conocen los patrones de tráfico. Con el modo de capacidad bajo demanda, usted paga por solicitud por las lecturas y escrituras de datos que la aplicación realiza en las tablas. No necesita especificar el rendimiento de lectura y escritura que espera de la aplicación, ya que DynamoDB se adapta de forma instantánea a las cargas de trabajo a medida que aumentan o disminuyen.

Cómo abordar las cargas de trabajo con aplicaciones enlazadas

En este escenario, la aplicación depende de otros sistemas, como los escenarios de procesamiento por lotes, en los que se pueden producir grandes picos de tráfico en función de los eventos de la lógica de la aplicación.

Considere desarrollar una lógica de escalado automática personalizada que reaccione ante aquellos eventos en los que pueda aumentar la capacidad de la tabla y `TargetValues` en función de las necesidades específicas. Podría aprovechar Amazon EventBridge y utilizar una combinación de servicios de AWS, como Lambda y Step Functions, para responder a las necesidades específicas de la aplicación.

Evaluación de la selección de clases de tabla de DynamoDB

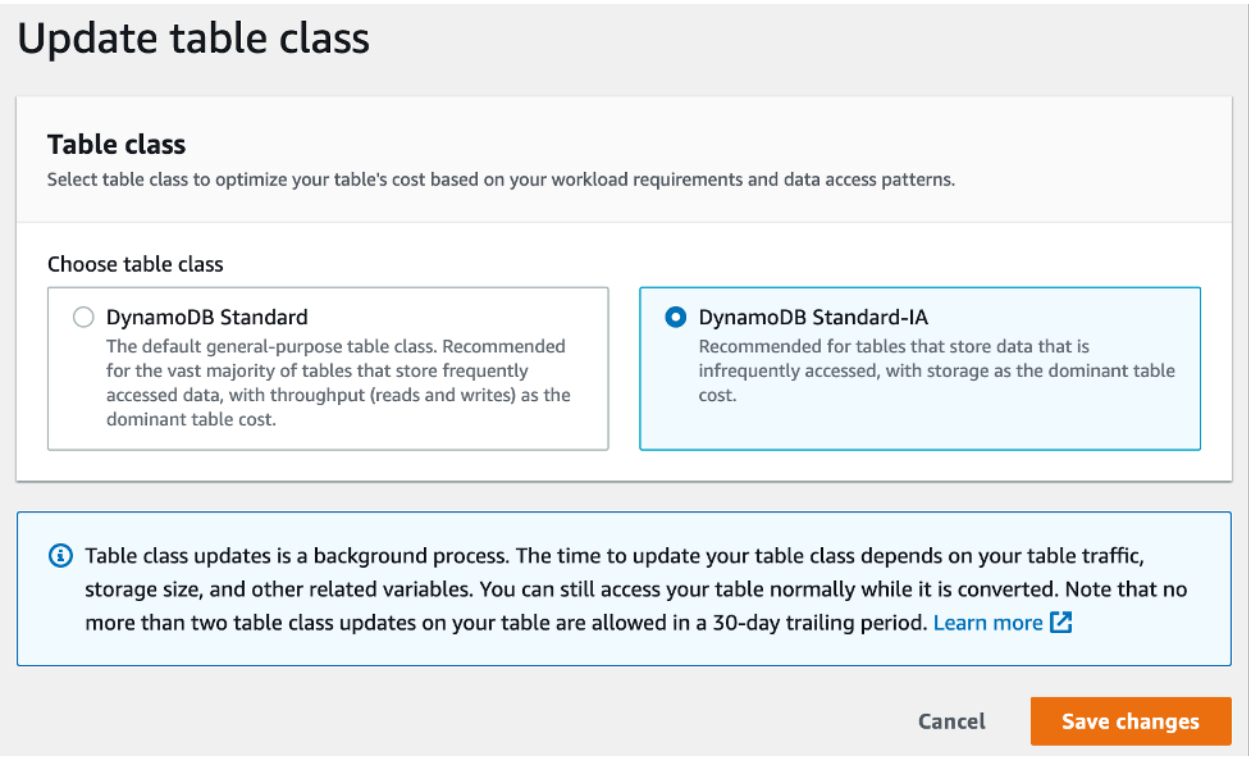
En esta sección se ofrece una visión general de cómo seleccionar la clase de tabla adecuada para su tabla de DynamoDB. Con el lanzamiento de la clase de tabla Estándar - Acceso poco frecuente estándar, ahora tiene la posibilidad de optimizar una tabla para obtener un menor costo de almacenamiento o un menor costo de rendimiento.

Temas

- Qué clases de tabla hay disponibles
- Cuándo seleccionar la clase de tabla de DynamoDB Standard
- Cuándo seleccionar la clase de tabla de DynamoDB Standard-IA
- Factores adicionales que se deben tener en cuenta al elegir una clase de tabla

Qué clases de tabla hay disponibles

Cuando cree una tabla de DynamoDB, debe seleccionar DynamoDB Standard o DynamoDB Standard-IA para la clase de tabla. La clase de tabla puede cambiarse dos veces en un periodo de 30 días, por lo que siempre podrá cambiarla en el futuro. La selección de una u otra clase de tabla no tiene ningún efecto sobre el rendimiento, la disponibilidad, la fiabilidad o la durabilidad de la tabla.



Clase de tabla estándar

La clase de tabla Estándar es la opción predeterminada para las tablas nuevas. Esta opción mantiene el equilibrio de facturación original de DynamoDB, que ofrece un equilibrio entre el rendimiento y los costos de almacenamiento para las tablas con datos de acceso frecuente.

Clase de tabla Estándar

La clase de tabla Estándar - Acceso poco frecuente ofrece un menor costo de almacenamiento (~60 % menos) para las cargas de trabajo que requieren un almacenamiento a largo plazo de los datos con actualizaciones o lecturas poco frecuentes. Dado que la clase está optimizada para un acceso poco frecuente, las lecturas y escrituras se facturarán a un costo ligeramente superior (~25 % más alto) que la clase de tabla Estándar.

Cuándo seleccionar la clase de tabla de DynamoDB Standard

La clase de tabla DynamoDB Standard es la más adecuada para las tablas cuyo costo de almacenamiento es aproximadamente el 50 % o menos del costo mensual total de la tabla. Este equilibrio de costos es indicativo de una carga de trabajo que accede o actualiza periódicamente elementos ya almacenados en DynamoDB.

Cuándo seleccionar la clase de tabla de DynamoDB Standard-IA

La clase de tabla DynamoDB Standard-IA es la más adecuada para las tablas cuyo costo de almacenamiento es aproximadamente el 50 % o más del costo mensual total de la tabla. Este equilibrio de costos es indicativo de una carga de trabajo que crea o lee menos elementos al mes de los que mantiene en el almacenamiento.

Un uso habitual de la clase de tablas Standard-IA es trasladar los datos a los que se accede con menos frecuencia a tablas Standard-IA individuales. Para obtener más información, consulte Optimizing the storage costs of your workloads with Amazon DynamoDB Standard-IA table class (Optimización de los costos de almacenamiento de sus cargas de trabajo con la clase de tabla Standard-IA de Amazon DynamoDB).

Factores adicionales que se deben tener en cuenta al elegir una clase de tabla

Al decidir entre las dos clases de tabla, hay algunos factores adicionales que vale la pena considerar como parte de la decisión.

Capacidad reservada

Actualmente no se admite la compra de capacidad reservada para las tablas que utilizan la clase de tabla Estándar - Acceso poco frecuente. Al pasar de una tabla Estándar con capacidad reservada a una tabla Estándar - Acceso poco frecuente sin capacidad reservada, es posible que no vea un beneficio de costo.

Identificación de recursos no utilizados en DynamoDB

En esta sección se ofrece una visión general de cómo evaluar periódicamente sus recursos sin utilizar. A medida que evolucionan los requisitos de su aplicación, debe asegurarse de que no haya recursos sin utilizar que contribuyan a generar costos de Amazon DynamoDB innecesarios. Los procedimientos descritos a continuación utilizarán las métricas de Amazon CloudWatch para identificar los recursos sin utilizar y le ayudarán a identificar y tomar medidas sobre esos recursos para reducir los costos.

Puede supervisar DynamoDB con CloudWatch, que recopila y procesa los datos sin procesar de DynamoDB y los convierte en métricas legibles casi en tiempo real. Estas estadísticas se conservan durante un periodo de tiempo, de modo que pueda acceder a la información histórica para comprender mejor su utilización. De forma predeterminada, los datos de las métricas de DynamoDB se envían a CloudWatch automáticamente. Para obtener más información, consulte ¿Qué es Amazon CloudWatch? y Retención de métricas en la Guía del usuario de Amazon CloudWatch.

Temas

- Cómo identificar los recursos sin utilizar
- Identificación de recursos de tabla sin utilizar
- Limpieza de los recursos de tabla no utilizados
- Identificación de recursos de GSI sin utilizar
- Limpieza de los recursos de GSI no utilizados
- Limpieza de tablas globales no utilizadas
- Limpieza de las copias de seguridad no utilizadas o recuperación en un momento dado (PITR)

Cómo identificar los recursos sin utilizar

Para identificar las tablas o los índices sin utilizar, examinaremos las siguientes métricas de CloudWatch a lo largo de un periodo de 30 días para saber si hay alguna lectura o escritura activa en la tabla o alguna lectura en los índices secundarios globales (GSI):

ConsumedReadCapacityUnits

Número de unidades de capacidad de lectura consumidas durante el periodo de tiempo especificado, para que pueda hacer un seguimiento de la capacidad consumida. Puede recuperar la capacidad total de lectura consumida para una tabla y todos sus índices secundarios globales, o para un índice secundario global determinado.

ConsumedWriteCapacityUnits

Número de unidades de capacidad de escritura consumidas durante el periodo de tiempo especificado, para que pueda hacer un seguimiento de la capacidad consumida. Puede recuperar la capacidad total de escritura consumida para una tabla y todos sus índices secundarios globales, o para un índice secundario global determinado.

Identificación de recursos de tabla sin utilizar

Amazon CloudWatch es un servicio de supervisión y observabilidad que proporciona las métricas de tabla de DynamoDB que utilizará para identificar los recursos sin utilizar. Las métricas de CloudWatch se pueden consultar mediante la Consola de administración de AWS y la AWS Command Line Interface.

AWS Command Line Interface

Para ver las métricas de sus tablas a través de la AWS Command Line Interface, puede utilizar los siguientes comandos.

1. En primer lugar, evalúe las lecturas de su tabla:

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedReadCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
```

   Para evitar la identificación falsa de una tabla como no utilizada, evalúe las métricas durante un periodo más largo. Elija un intervalo de tiempo de inicio y fin apropiado, como 30 días, y un periodo apropiado, como 86400.

   En los datos devueltos, cualquier suma superior a 0 indica que la tabla que está evaluando recibió tráfico de lectura durante ese periodo.

   En el siguiente resultado se muestra una tabla que recibe tráfico de lectura en el periodo evaluado:

```
        {
            "Timestamp": "2022-08-25T19:40:00Z",
            "Sum": 36023355.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-12T19:40:00Z",
            "Sum": 38025777.5,
            "Unit": "Count"
        },
```

   En el siguiente resultado se muestra una tabla que no recibe tráfico de lectura en el periodo evaluado:

```
        {
            "Timestamp": "2022-08-01T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

```
        {
            "Timestamp": "2022-08-20T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

2. A continuación, evalúe las escrituras de su tabla:

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedWriteCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
```

Para evitar la identificación falsa de una tabla como no utilizada, es recomendable evaluar las métricas durante un periodo más largo. Elija un intervalo de tiempo de inicio y de finalización apropiado, como 30 días, y un periodo apropiado, como 86 400.

En los datos devueltos, cualquier suma superior a 0 indica que la tabla que está evaluando recibió tráfico de lectura durante ese periodo.

En el siguiente resultado se muestra una tabla que recibe el tráfico de escritura en el periodo evaluado:

```
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 41014457.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-18T20:15:00Z",
            "Sum": 40048531.0,
            "Unit": "Count"
        },
```

En el siguiente resultado se muestra una tabla que no recibe tráfico de escritura en el periodo evaluado:

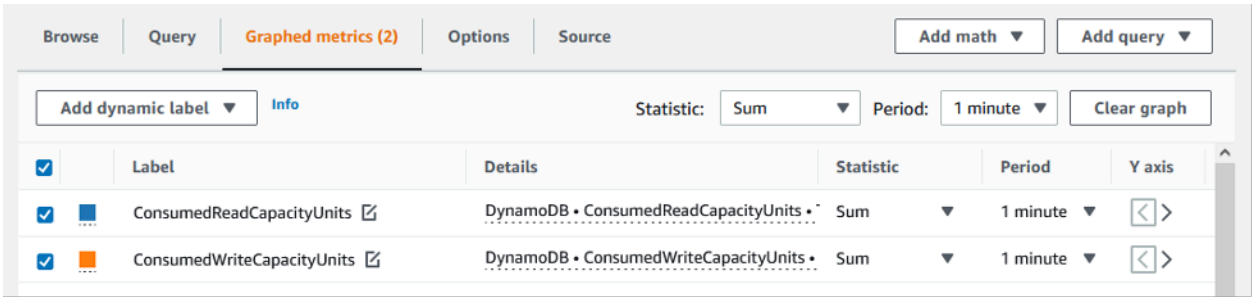
```
        {
            "Timestamp": "2022-07-31T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
```

```
        },
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

Consola de administración de AWS

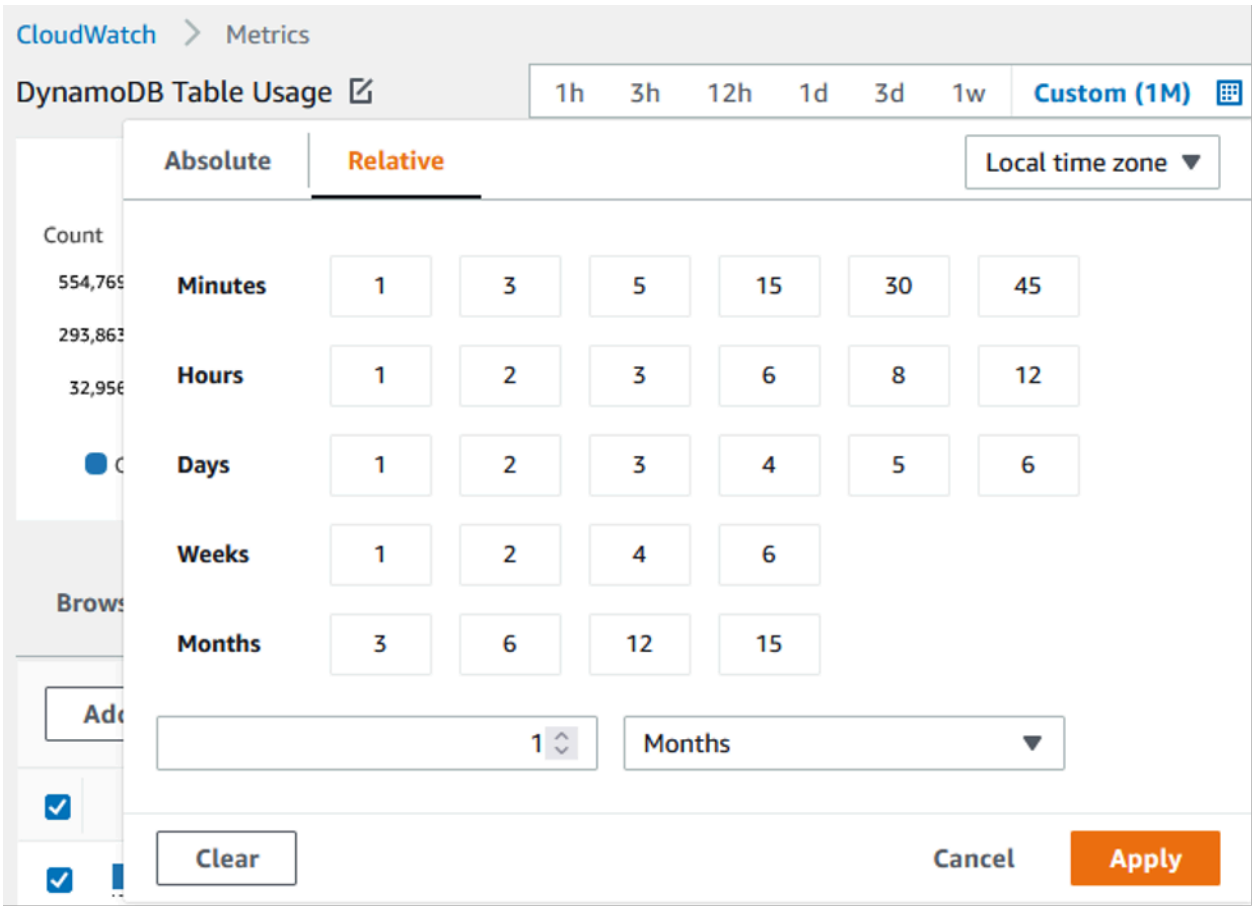
Los siguientes pasos le permitirán evaluar la utilización de sus recursos a través de la Consola de administración de AWS.

1. Acceda a la consola de AWS y navegue hasta la página del servicio CloudWatch en https://console.aws.amazon.com/cloudwatch/. Seleccione la región de AWS correspondiente en la parte superior derecha de la consola, si es necesario.
2. En el panel de navegación izquierdo, busque la sección Metrics (Métricas) y seleccione All metrics (Todas las métricas).
3. La acción anterior abrirá un panel de control con dos paneles. En el panel superior verá las métricas representadas actualmente. En la parte inferior seleccionará las métricas disponibles para representarlas. Seleccione DynamoDB en el panel inferior.
4. En el panel de selección de métricas de DynamoDB, seleccione la categoría Table Metrics (Métricas de tabla) para mostrar las métricas de sus tablas en la región actual.
5. Identifique el nombre de su tabla desplazándose hacia abajo en el menú. A continuación, seleccione las métricas `ConsumedReadCapacityUnits` y `ConsumedWriteCapacityUnits` para su tabla.
6. Seleccione la pestaña Graphed metrics (2) (Métricas diagramadas [2]) y ajuste la columna Statistic (Estadística) a Sum (Suma).



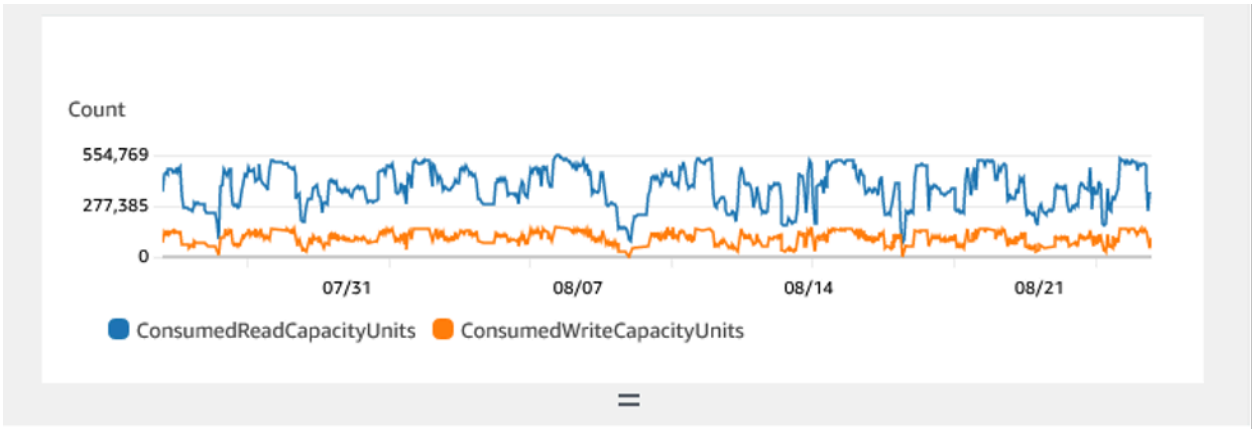
7. Para evitar la identificación falsa de una tabla como no utilizada, es recomendable evaluar las métricas durante un periodo más largo. En la parte superior del panel de gráficos, elija un

periodo de tiempo adecuado, por ejemplo, un mes, para evaluar su tabla. Seleccione Custom (Personalizado) y 1 Months (1 mes) en las listas desplegables. Elija Apply (Aplicar).

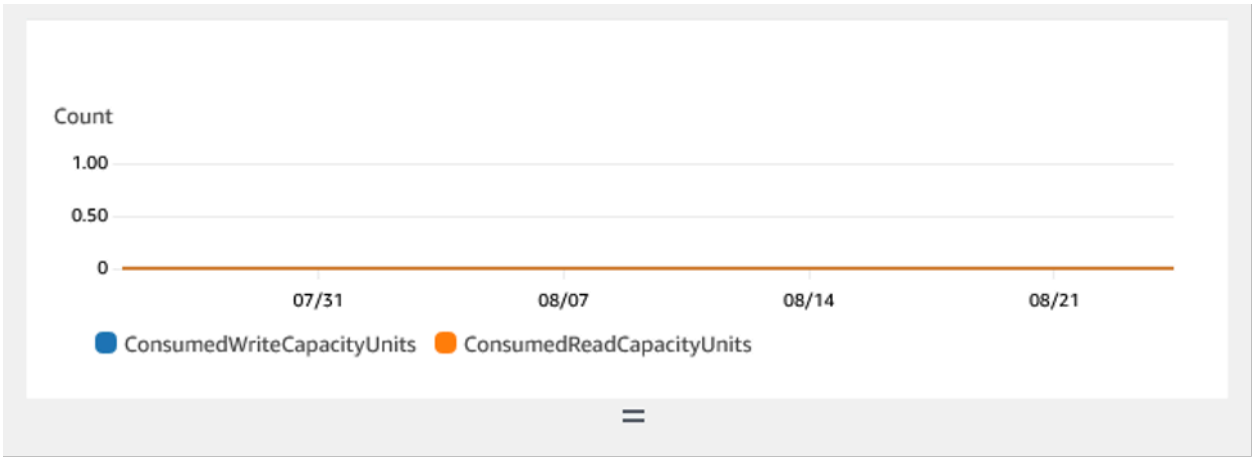


8. Evalúe la métrica diagramada de su tabla para determinar si se está utilizando. Las métricas que han subido por encima de 0 indican que se ha utilizado una tabla durante el periodo de tiempo evaluado. Un gráfico plano a 0 tanto para la lectura como para la escritura indica una tabla que no se utiliza.

   En la siguiente imagen se muestra una tabla con tráfico de lectura:



En la siguiente imagen se muestra una tabla sin el tráfico de lectura:



Limpieza de los recursos de tabla no utilizados

Si ha identificado los recursos de tabla no utilizados, puede reducir sus costos continuos de las siguientes maneras.

> **Note**
>
> Si ha identificado una tabla que no se utiliza, pero quiere mantenerla disponible por si es necesario acceder a ella en el futuro, considere la posibilidad de cambiarla al modo bajo demanda. De lo contrario, puede plantearse hacer una copia de seguridad y eliminar la tabla por completo.

Modos de capacidad

DynamoDB cobra por leer, escribir y almacenar datos en sus tablas de DynamoDB.

DynamoDB tiene dos modos de capacidad, que incorporan opciones de facturación específicas para el procesamiento de lecturas y escrituras en sus tablas: bajo demanda y aprovisionada. El modo de capacidad de lectura/escritura controla cómo se le cobrará el rendimiento de lectura y escritura y cómo se administra la capacidad.

Para tablas en modo en diferido, no necesita especificar el rendimiento de lectura y escritura que espera de su aplicación. DynamoDB le carga las lecturas y escrituras que la aplicación lleva a cabo en las tablas en términos de unidades de solicitud de lectura y de escritura, respectivamente. Si no

hay actividad en su tabla o índice, no pagará por el rendimiento, pero sí incurrirá en un cargo por almacenamiento.

Clase de tabla

DynamoDB también ofrece dos clases de tablas diseñadas para ayudarle a optimizar el costo. La clase de tabla DynamoDB Standard es la predeterminada y se recomienda para la mayoría de las cargas de trabajo. La clase de tabla DynamoDB Estándar - Acceso poco frecuente (DynamoDB Standard-IA) está optimizada para tablas en las que el almacenamiento es el costo dominante.

Si no hay actividad en la tabla o el índice, es probable que el almacenamiento sea el costo dominante y el cambio de clase de tabla ofrecerá un ahorro significativo.

Eliminación de tablas

Si ha descubierto una tabla que no se utiliza y desea eliminarla, es recomendable hacer primero una copia de seguridad o exportar los datos.

Las copias de seguridad realizadas a través de AWS Backup pueden aprovechar el almacenamiento en frío por niveles, lo que reduce aún más los costos. Consulte la documentación de Uso de AWS Backup con DynamoDB para obtener información sobre cómo habilitar las copias de seguridad a través de AWS Backup, así como la documentación de Managing backup plans (Administración de los planes de copia de seguridad) para obtener información sobre cómo utilizar el ciclo de vida para mover la copia de seguridad al almacenamiento en frío.

Como alternativa, puede exportar los datos de la tabla a S3. Para ello, consulte la documentación de Export to Amazon S3 (Exportación a Amazon S3). Una vez exportados los datos, si desea aprovechar S3 Glacier Instant Retrieval, S3 Glacier Flexile Retrieval o S3 Glacier Deep Archive para reducir aún más los costos, consulte Administración del ciclo de vida del almacenamiento.

Una vez realizada la copia de seguridad de la tabla, puede eliminarla a través de la Consola de administración de AWS o la AWS Command Line Interface.

Identificación de recursos de GSI sin utilizar

Los pasos para identificar un índice secundario global no utilizado son similares a los de la identificación de una tabla no utilizada. Como DynamoDB replica los elementos escritos en su tabla base en su GSI si contienen el atributo utilizado como clave de partición del GSI, es probable que un GSI no utilizado siga teniendo `ConsumedWriteCapacityUnits` por encima de 0 si su tabla

base está en uso. Como resultado, solo evaluará la métrica `ConsumedReadCapacityUnits` para determinar si su GSI no se utiliza.

Para ver las métricas de sus GSI a través de la AWS CLI de AWS, puede utilizar los siguientes comandos para evaluar las lecturas de las tablas:

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedReadCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/DynamoDB --statistics Sum --
dimensions Name=TableName,Value=<table-name>
Name=GlobalSecondaryIndexName,Value=<index-name>
```

Para evitar la identificación falsa de una tabla como no utilizada, es recomendable evaluar las métricas durante un periodo más largo. Elija un intervalo de tiempo de inicio y de finalización apropiado, como 30 días, y un periodo apropiado, como 86400.

En los datos devueltos, cualquier suma superior a 0 indica que la tabla que está evaluando recibió tráfico de lectura durante ese periodo.

En el siguiente resultado se muestra un GSI que recibe tráfico de lectura en el periodo evaluado:

```
        {
          "Timestamp": "2022-08-17T21:20:00Z",
          "Sum": 36319167.0,
          "Unit": "Count"
        },
        {
          "Timestamp": "2022-08-11T21:20:00Z",
          "Sum": 1869136.0,
          "Unit": "Count"
        },
```

En el siguiente resultado se muestra un GSI que recibe tráfico de lectura mínimo en el periodo evaluado:

```
        {
          "Timestamp": "2022-08-28T21:20:00Z",
          "Sum": 0.0,
          "Unit": "Count"
        },
        {
```

```
      "Timestamp": "2022-08-15T21:20:00Z",
      "Sum": 2.0,
      "Unit": "Count"
    },
```

En el siguiente resultado se muestra un GSI que no recibe tráfico de lectura en el periodo evaluado:

```
    {
      "Timestamp": "2022-08-17T21:20:00Z",
      "Sum": 0.0,
      "Unit": "Count"
    },
    {
      "Timestamp": "2022-08-11T21:20:00Z",
      "Sum": 0.0,
      "Unit": "Count"
    },
```

Limpieza de los recursos de GSI no utilizados

Si ha identificado un GSI no utilizado, puede eliminarlo. Como todos los datos presentes en un GSI están también presentes en la tabla base, no es necesario realizar una copia de seguridad adicional antes de eliminar un GSI. Si en el futuro se vuelve a necesitar el GSI, se puede volver a agregar a la tabla.

Si ha identificado un GSI de uso poco frecuente, debería considerar la posibilidad de realizar cambios de diseño en su aplicación que le permitan eliminarlo o reducir su costo. Por ejemplo, aunque los escaneos de DynamoDB deben utilizarse con moderación porque pueden consumir grandes cantidades de recursos del sistema, pueden ser más rentables que un GSI si el patrón de acceso que admite se utiliza con muy poca frecuencia.

Además, si se requiere un GSI para respaldar un patrón de acceso poco frecuente, considere la posibilidad de proyectar un conjunto más limitado de atributos. Aunque esto puede requerir consultas posteriores en la tabla base para respaldar sus patrones de acceso poco frecuentes, puede ofrecer potencialmente una reducción significativa de los costos de almacenamiento y escritura.

Limpieza de tablas globales no utilizadas

Las tablas globales de Amazon DynamoDB proporcionan una solución completamente administrada para implementar una base de datos en varias regiones y multiactiva sin tener que crear ni mantener su propia solución de replicación.

Las tablas globales son ideales para proporcionar un acceso de baja latencia a los datos cerca de los usuarios y también como región secundaria para la recuperación de desastres.

Si la opción de tablas generales está activada para un recurso con el fin de proporcionar un acceso de baja latencia a los datos, pero no forma parte de su estrategia de recuperación de desastres, valide que ambas réplicas están sirviendo activamente el tráfico de lectura mediante la evaluación de sus métricas de CloudWatch. Si una réplica no sirve tráfico de lectura, puede ser un recurso no utilizado.

Si las tablas generales forman parte de su estrategia de recuperación de desastres, puede esperarse que una réplica no reciba tráfico de lectura según un patrón activo/en espera.

Limpieza de las copias de seguridad no utilizadas o recuperación en un momento dado (PITR)

DynamoDB ofrece dos estilos de copia de seguridad. La recuperación en un momento dado proporciona copias de seguridad continuas hasta 35 días para ayudarlo a protegerse contra escrituras o eliminaciones accidentales, mientras que la copia de seguridad bajo demanda permite la creación de instantáneas que pueden guardarse a largo plazo. Puede establecer el periodo de recuperación en cualquier valor entre 1 y 35 días. Ambos tipos de copias de seguridad tienen costos asociados.

Consulte la documentación de Copia de seguridad y restauración para DynamoDB y Copias de seguridad a un momento dado para DynamoDB para determinar si sus tablas tienen activadas copias de seguridad que ya no sean necesarias.

Evaluación de los patrones de uso de las tablas de DynamoDB

En esta sección se proporciona información general de cómo evaluar si está usando las tablas de DynamoDB de forma eficiente. Hay determinados patrones de uso que no son óptimos para DynamoDB y permiten la optimización tanto desde el punto de vista del rendimiento como de los costos.

Temas

- Realizar menos operaciones de lectura altamente coherente
- Realizar menos transacciones para las operaciones de lectura
- Realizar menos operaciones de análisis
- Acortamiento de nombres de atributos
- Habilitar el Periodo de vida (TTL)
- Reemplazar las tablas globales por copias de seguridad entre regiones

Realizar menos operaciones de lectura altamente coherente

DynamoDB le permite configurar la coherencia de lectura por solicitud. Las solicitudes de lectura tienen coherencia final de forma predeterminada. Las lecturas coherentes posteriores se cobran a 0,5 RCU para un máximo de 4 KB de datos.

La mayoría de las partes de las cargas de trabajo distribuidas son flexibles y pueden tolerar una coherencia eventual. Sin embargo, puede haber patrones de acceso que requieran lecturas altamente coherentes. Las lecturas altamente coherentes se cobran a 1 RCU por hasta 4 KB de datos, lo que básicamente duplica los costos de lectura. DynamoDB le ofrece la flexibilidad de utilizar ambos modelos de coherencia en la misma tabla.

Puede evaluar la carga de trabajo y el código de la aplicación para confirmar si solo se utilizan lecturas altamente coherentes cuando sea necesario.

Realizar menos transacciones para las operaciones de lectura

DynamoDB le permite agrupar determinadas acciones en forma de "todo o nada", lo que significa que puede ejecutar transacciones ACID con DynamoDB. Sin embargo, como ocurre con las bases de datos relacionales, no es necesario seguir este enfoque para cada acción.

Una operación de lectura transaccional de hasta 4 KB consume 2 RCU, a diferencia de las 0,5 RCU predeterminadas, para leer la misma cantidad de datos de manera eventualmente coherente. Los costos se duplican en las operaciones de escritura, lo que significa que una escritura transaccional de hasta 1 KB equivale a 2 WCU.

Para determinar si todas las operaciones de las tablas son transacciones, las métricas de CloudWatch de la tabla se pueden filtrar hasta las API de transacciones. Si las API de transacciones son los únicos gráficos disponibles en las métricas `SuccessfulRequestLatency` de la tabla, esto confirmaría que cada operación es una transacción para esta tabla. Como alternativa, si la tendencia de utilización de la capacidad general coincide con la tendencia de la API de transacciones, considere revisitar el diseño de la aplicación, ya que parece estar dominado por las API transaccionales.

Realizar menos operaciones de análisis

El uso generalizado de las operaciones de `Scan` generalmente se debe a la necesidad de ejecutar consultas analíticas en los datos de DynamoDB. Realizar operaciones de `Scan` frecuentes en una tabla grande puede resultar ineficiente y costoso.

Una mejor alternativa es utilizar la función Exportar a S3 y elegir un punto en el tiempo para exportar el estado de la tabla a S3. AWS ofrece servicios como Athena, que luego se pueden utilizar para ejecutar consultas analíticas sobre los datos sin consumir ninguna capacidad de la tabla.

La frecuencia de las operaciones de `Scan` se puede determinar utilizando la estadística de `SampleCount` situada en la métrica `SuccessfulRequestLatency` para `Scan`. Si las operaciones de `Scan` son realmente muy frecuentes, se deben volver a evaluar los patrones de acceso y el modelo de datos.

Acortamiento de nombres de atributos

El tamaño total de un elemento en DynamoDB es la suma de las longitudes y los valores de los nombres de los atributos. Tener nombres de atributos largos no solo contribuye a los costos de almacenamiento, sino que es posible que también aumente el consumo de RCU o WCU. Recomendamos elegir nombres de atributos cortos en lugar de largos. Tener nombres de atributos más cortos puede ayudar a limitar el tamaño del elemento dentro del siguiente límite de 4 KB/1 KB, después del cual consumiría RCU o WCU adicionales para acceder a los datos.

Habilitar el Periodo de vida (TTL)

El Periodo de vida (TTL) puede identificar los elementos que tengan una antigüedad superior a la fecha de caducidad que ha establecido en un artículo y eliminarlos de la tabla. Si los datos aumentan con el tiempo y los datos más antiguos se vuelven irrelevantes, habilitar el TTL en la tabla puede ayudar a reducir los datos y ahorrar en costos de almacenamiento.

Otro aspecto útil del TTL es que los elementos caducados aparecen en las transmisiones de DynamoDB, por lo que, en lugar de limitarse a eliminar los datos de los datos, es posible consumir esos elementos de la transmisión y archivarlos en un nivel de almacenamiento más económico. Además, eliminar elementos mediante TTL no supone ningún costo adicional, ya que no consume capacidad y el diseño de una aplicación de limpieza no implica gastos adicionales.

Reemplazar las tablas globales por copias de seguridad entre regiones

Las tablas globales le permiten mantener varias tablas de réplica activas en diferentes regiones; todas pueden aceptar operaciones de escritura y replicar datos entre sí. Es fácil configurar las réplicas y la sincronización se administra automáticamente. Las réplicas convergen en un estado coherente mediante la estrategia de "el último escritor gana".

Si utiliza tablas globales únicamente como parte de una estrategia de conmutación por error o recuperación de desastres (DR), puede sustituirlas por una copia de seguridad entre regiones para cumplir objetivos de puntos de recuperación y objetivos de tiempo de recuperación relativamente

flexibles. Si no necesita un acceso local rápido y una alta disponibilidad de cinco nueves, es posible que mantener una réplica de tabla global no sea el mejor enfoque para la conmutación por error.

Como alternativa, considere utilizar AWS Backup para administrar las copias de seguridad de DynamoDB. Puede programar copias de seguridad periódicas y copiarlas en todas las regiones para cumplir con los requisitos de DR con un enfoque más rentable en comparación con el uso de tablas globales.

Evaluación del uso de los flujos de DynamoDB

En esta sección se proporciona información general sobre cómo evaluar el uso de DynamoDB Streams. Hay determinados patrones de uso que no son óptimos para DynamoDB y permiten la optimización tanto desde el punto de vista del rendimiento como de los costos.

Tiene dos integraciones de streaming nativas para el streaming y los casos de uso basados en eventos:

- Amazon DynamoDB Streams
- Amazon Kinesis Data Streams

Esta página solo se centrará en las estrategias de optimización de costos para estas opciones. Si, en cambio, quiere saber cómo elegir entre las dos opciones, consulte Opciones de streaming para la captura de datos de cambio.

Temas

- Optimización de costos para DynamoDB Streams
- Optimización de los costos para Kinesis Data Streams
- Estrategias de optimización de costos para ambos tipos de uso de Streams

Optimización de costos para DynamoDB Streams

Como se indica en la página de precios de DynamoDB Streams, independientemente del modo de capacidad de rendimiento de la tabla, DynamoDB cobra según la cantidad de solicitudes de lectura realizadas para DynamoDB Stream de la tabla. Las solicitudes de lectura realizadas a DynamoDB Stream son diferentes de las solicitudes de lectura realizadas a una tabla de DynamoDB.

Cada solicitud de lectura en términos de flujo tiene la forma de una llamada a la API de `GetRecords` que puede devolver hasta 1000 registros o 1 MB de registros en la respuesta, lo que se alcance

primero. Ninguna de las demás API de DynamoDB Stream genera gastos y DynamoDB Streams no genera gastos por estar inactivo. En otras palabras, si no se realiza ninguna solicitud de lectura a DynamoDB Stream, no se generará ningún gasto por tener DynamoDB Stream habilitado en una tabla.

Estas son algunas de las aplicaciones de consumidor para DynamoDB Streams:

- Funciones de AWS Lambda
- Aplicaciones basadas en Amazon Kinesis Data Streams
- Aplicaciones para clientes y consumidores creadas con un AWS SDK

Las solicitudes de lectura realizadas por los consumidores de DynamoDB Streams basadas en AWS Lambda son gratuitas, mientras que las llamadas realizadas por consumidores de cualquier otro tipo tienen un costo. Cada mes, las primeras 2 500 000 de solicitudes de lectura realizadas por consumidores que no son de Lambda también son gratuitas. Esto se aplica a todas las solicitudes de lectura realizadas a cualquier DynamoDB Streams en una cuenta de AWS para cada región de AWS.

Monitoreo del uso de DynamoDB Streams

Los cargos de DynamoDB Streams en la consola de facturación se agrupan para todas las DynamoDB Streams de la región de AWS en una cuenta de AWS. Actualmente, no se admite el etiquetado de DynamoDB Streams, por lo que las etiquetas de asignación de costos no se pueden utilizar para identificar los costos precisos de DynamoDB Streams. El volumen de llamadas de `GetRecords` se puede obtener en el nivel de DynamoDB Streams para calcular los cargos por transmisión. El volumen se representa mediante la métrica de CloudWatch de DynamoDB Stream `SuccessfulRequestLatency` y la estadística de `SampleCount`. Esta métrica también incluirá las llamadas de `GetRecords` realizadas por tablas globales para realizar una replicación continua, así como las llamadas realizadas por los consumidores de AWS Lambda, las cuales no se cobran. Para obtener información sobre otras métricas de CloudWatch publicadas por DynamoDB Streams, consulte Dimensiones y métricas de DynamoDB.

Uso de AWS Lambda como consumidor

Evalúe si es factible utilizar funciones de AWS Lambda como consumidores para DynamoDB Streams, ya que así se pueden eliminar los costos asociados a la lectura de DynamoDB Streams. Por otro lado, a las aplicaciones de consumidores basadas en el SDK o el adaptador Kinesis de DynamoDB Streams se les cobrará en función del número de llamadas de `GetRecords` que realicen a DynamoDB Stream.

Las invocaciones de funciones de Lambda se cobrarán según el precio estándar de Lambda; sin embargo, DynamoDB Streams no incurrirá en ningún cargo. Lambda sondeará las particiones de DynamoDB Stream y buscará registros 4 veces por segundo. Cuando hay registros disponibles, Lambda invoca la función y espera el resultado. Si el procesamiento se realiza correctamente, Lambda reanuda el sondeo hasta que recibe más registros.

Ajuste de las aplicaciones de consumidor basadas en el adaptador Kinesis de DynamoDB Streams

Dado que las solicitudes de lectura realizadas por consumidores que no utilizan Lambda se cobran por DynamoDB Streams, es importante encontrar un equilibrio entre el requisito casi en tiempo real y la cantidad de veces que la aplicación del consumidor debe sondear la DynamoDB Stream.

La frecuencia de sondeo de DynamoDB Streams mediante una aplicación basada en el adaptador Kinesis de DynamoDB Streams viene determinada por el valor de `idleTimeBetweenReadsInMillis` configurado. Este parámetro determina la cantidad de tiempo en milisegundos que el consumidor debe esperar antes de procesar una partición en caso de que la llamada de `GetRecords` anterior realizada a la misma partición no devolviera ningún registro. De forma predeterminada, este valor para este parámetro es 1000 ms. Si no se requiere procesamiento casi en tiempo real, este parámetro podría aumentar para que la aplicación del consumidor realice menos llamadas de `GetRecords` y optimice las llamadas de DynamoDB Streams.

Optimización de los costos para Kinesis Data Streams

Cuando se establece Kinesis Data Stream como destino para entregar eventos de captura de datos de cambios para una tabla de DynamoDB, es posible que Kinesis Data Stream necesite una administración de tamaños independiente, lo que afectará a los costos generales. DynamoDB cobra en términos de unidades de captura de datos de cambio (CDU), donde cada unidad se compone de un elemento de DynamoDB de tamaño máximo de 1 KB que el servicio de DynamoDB intenta enviar a Kinesis Data Stream de destino.

Además de los cargos del servicio de DynamoDB, se cobrarán los cargos estándar de Kinesis Data Stream. En la página de precios se indica que los del servicio varían en función del modo de capacidad (aprovisionado y bajo demanda), que son distintos de los modos de capacidad de las tablas de DynamoDB y están definidos por el usuario. A un nivel alto, Kinesis Data Streams cobra una tarifa por hora en función del modo de capacidad y de la ingesta de datos en la transmisión del servicio de DynamoDB. Es posible que se impongan cargos adicionales como la recuperación de datos (para el modo bajo demanda), la retención de datos extendida (más allá de las 24 horas predeterminadas) y la mejora de las recuperaciones generalizada por parte de los consumidores, según la configuración del usuario para Kinesis Data Stream.

Monitoreo del uso de Kinesis Data Streams

Kinesis Data Streams para DynamoDB publica métricas de DynamoDB además de las métricas estándar de Kinesis Data Stream CloudWatch. Es posible que el servicio de Kinesis restrinja un intento `Put` del servicio de DynamoDB debido a que la capacidad de Kinesis Data Streams es insuficiente o que lo hagan componentes dependientes, como un servicio de AWS KMS que puede estar configurado para cifrar los datos en reposo de Kinesis Data Stream.

Para obtener más información sobre las métricas de CloudWatch publicadas por el servicio de DynamoDB para Kinesis Data Stream, consulte Monitoreo de la captura de datos de cambios con Kinesis Data Streams. Para evitar costos adicionales derivados de los reintentos del servicio debido a las limitaciones, es importante dimensionar correctamente Kinesis Data Stream en el caso del modo aprovisionado.

Cómo elegir el modo de capacidad adecuado para Kinesis Data Streams

Kinesis Data Streams admite dos modos de capacidad: modo aprovisionado y modo bajo demanda.

- Si la carga de trabajo que involucra a Kinesis Data Stream tiene un tráfico de aplicaciones predecible, un tráfico constante o aumenta gradualmente o un tráfico que se puede prever con precisión, entonces el modo aprovisionado de Kinesis Data Streams es adecuado y será más rentable
- Si la carga de trabajo es nueva, tiene un tráfico de aplicaciones impredecible o si prefiere no administrar la capacidad, el modo bajo demanda de Kinesis Data Streams es adecuado y será más rentable

Una práctica recomendada para optimizar los costos sería evaluar si la tabla de DynamoDB asociada a Kinesis Data Stream tiene un patrón de tráfico predecible que pueda aprovechar el modo aprovisionado de Kinesis Data Streams. Si la carga de trabajo es nueva, puede utilizar el modo bajo demanda para Kinesis Data Streams durante unas semanas de inicio, revisar las métricas de CloudWatch para comprender los patrones de tráfico y, a continuación, cambiar la misma transmisión al modo aprovisionado en función de la naturaleza de la carga de trabajo. En el caso del modo aprovisionado, la estimación de las particiones numéricas se puede realizar siguiendo las consideraciones de administración de particiones de Kinesis Data Streams.

Evaluar las aplicaciones del consumidor con Kinesis Data Streams para DynamoDB

Como Kinesis Data Streams no cobra por la cantidad de llamadas de `GetRecords`, como DynamoDB Streams, las aplicaciones de consumidor pueden realizar la mayor cantidad de llamadas

posible, siempre que la frecuencia esté por debajo de la limitación para `GetRecords`. En cuanto al modo bajo demanda de Kinesis Data Streams, las lecturas de datos se cobran por GB. Para Kinesis Data Streams en modo aprovisionado, las lecturas no se cobran si los datos tienen menos de 7 días de antigüedad. Para funciones de Lambda como consumidores de Kinesis Data Streams, Lambda sondea cada partición de Kinesis Stream y busca registros una vez por segundo.

Estrategias de optimización de costos para ambos tipos de uso de Streams

Filtrado de eventos para consumidores de AWS Lambda

El filtrado de eventos de Lambda permite descartar los eventos en función de un criterio de filtro para que no estén disponibles en el lote de invocación de funciones de Lambda. Esto optimiza los costos de Lambda para procesar o descartar registros de flujo no deseados dentro de la lógica de funciones del consumidor. Para obtener más información sobre cómo configurar el filtrado de eventos y escribir los criterios de filtrado, consulte Filtrado de eventos de Lambda.

Ajuste de los consumidores de AWS Lambda

Los costos se podrían optimizar aún más ajustando los parámetros de configuración de Lambda, como aumentar el `BatchSize` para procesar más por invocación, permitir `BisectBatchOnFunctionError` para evitar el procesamiento de duplicados (lo que genera costos adicionales) y configurar `MaximumRetryAttempts` para no tener demasiados reintentos. De forma predeterminada, las invocaciones de Lambda para consumidores que producen error se vuelven a intentar indefinidamente hasta que el registro caduque de la transmisión, es decir, unas 24 horas en el caso de DynamoDB Streams y se puede configurar desde 24 horas hasta 1 año para Kinesis Data Streams. En la guía para desarrolladores de AWS Lambda encontrará opciones de configuración de Lambda adicionales disponibles, incluidas las mencionadas anteriormente para los consumidores de DynamoDB Stream.

Evaluación de la capacidad aprovisionada para lograr un aprovisionamiento del tamaño adecuado en la tabla de DynamoDB

En esta sección se proporciona información general sobre cómo evaluar si tiene aprovisionamiento del tamaño adecuado en las tablas de DynamoDB. A medida que evolucione la carga de trabajo, debe modificar los procedimientos operativos de manera adecuada, especialmente si la tabla de DynamoDB está configurada en modo aprovisionado y corre el riesgo de aprovisionar en exceso o de manera insuficiente las tablas.

Los procedimientos que se describen a continuación requieren información estadística que se debe capturar de las tablas de DynamoDB que son compatibles con la aplicación de producción.

Para entender el comportamiento de la aplicación, debe definir un periodo de tiempo que sea lo suficientemente significativo como para captar la estacionalidad de los datos de la aplicación. Por ejemplo, si la aplicación muestra patrones semanales, utilizar un periodo de tres semanas debería darle suficiente espacio para analizar las necesidades de rendimiento de la aplicación.

Si no sabe por dónde empezar, utilice al menos un mes de uso de datos para los cálculos que se indican a continuación.

Al evaluar la capacidad, las tablas de DynamoDB pueden configurar las unidades de capacidad de lectura (RCU) y las unidades de capacidad de escritura (WCU) de forma independiente. Si las tablas tienen algún índice secundario global (GSI) configurado, deberá especificar el rendimiento que consumirá, que también será independiente de las RCU y las WCU de la tabla base.

> **Note**
>
> Los índices secundarios locales (LSI) consumen capacidad de la tabla base.

Temas

- Cómo recuperar las métricas de consumo de las tablas de DynamoDB
- Cómo identificar tablas de DynamoDB con falta de aprovisionamiento
- Cómo identificar tablas de DynamoDB con exceso de aprovisionamiento

Cómo recuperar las métricas de consumo de las tablas de DynamoDB

Para evaluar la tabla y la capacidad del GSI, monitoree las siguientes métricas de CloudWatch y seleccione la dimensión adecuada para recuperar la información de la tabla o del GSI:

Unidades de capacidad de lectura	Unidades de capacidad de escritura
`ConsumedReadCapacityUnits`	`ConsumedWriteCapacityUnits`
`ProvisionedReadCapacityUnits`	`ProvisionedWriteCapacityUnits`
`ReadThrottleEvents`	`WriteThrottleEvents`

Puede hacer esto a través de la AWS CLI o la Consola de administración de AWS.

AWS CLI

Antes de recuperar las métricas de consumo de la tabla, tendremos que empezar por capturar algunos puntos de datos históricos mediante la API de CloudWatch.

Comience por crear dos archivos: `write-calc.json` y `read-calc.json`. Estos archivos representarán los cálculos de una tabla o GSI. Deberá actualizar algunos de los campos, como se indica en la tabla siguiente, para que coincidan con el entorno.

Nombre del campo	Definición	Ejemplo
`<table-name>`	El nombre de la tabla que analizará	SampleTable
`<period>`	El periodo de tiempo que utilizará para evaluar el objetivo de utilización, en segundos	Para un periodo de 1 hora, debe especificar: 3600
`<start-time>`	El comienzo del intervalo de evaluación, especificado en formato ISO8601	2022-02-21T23:00:00
`<end-time>`	El final del intervalo de evaluación, especificado en formato ISO8601	2022-02-22T06:00:00

El archivo de cálculos de escritura recuperará la cantidad de WCU aprovisionadas y consumidas en el periodo de tiempo del intervalo de fechas especificado. También generará un porcentaje de utilización que se usará para el análisis. El contenido completo del archivo `write-calc.json` debe ser similar al siguiente:

```
{
  "MetricDataQueries": [
    {
      "Id": "provisionedWCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
```

```
          "MetricName": "ProvisionedWriteCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Average"
      },
      "Label": "Provisioned",
      "ReturnData": false
    },
    {
      "Id": "consumedWCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ConsumedWriteCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>""
            }
          ]
        },
        "Period": <period>,
        "Stat": "Sum"
      },
      "Label": "",
      "ReturnData": false
    },
    {
      "Id": "m1",
      "Expression": "consumedWCU/PERIOD(consumedWCU)",
      "Label": "Consumed WCUs",
      "ReturnData": false
    },
    {
      "Id": "utilizationPercentage",
      "Expression": "100*(m1/provisionedWCU)",
      "Label": "Utilization Percentage",
      "ReturnData": true
```

```
    }
  ],
  "StartTime": "<start-time>",
  "EndTime": "<ent-time>",
  "ScanBy": "TimestampDescending",
  "MaxDatapoints": 24
}
```

El archivo de cálculos de lectura utiliza un archivo similar. Este archivo recuperará cuántas RCU se aprovisionaron y consumieron durante el periodo de tiempo del intervalo de fechas especificado. El contenido del archivo `read-calc.json` debe ser similar al siguiente:

```
{
  "MetricDataQueries": [
    {
      "Id": "provisionedRCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ProvisionedReadCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Average"
      },
      "Label": "Provisioned",
      "ReturnData": false
    },
    {
      "Id": "consumedRCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/DynamoDB",
          "MetricName": "ConsumedReadCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
```

```
            }
          ]
        },
        "Period": <period>,
        "Stat": "Sum"
      },
      "Label": "",
      "ReturnData": false
    },
    {
      "Id": "m1",
      "Expression": "consumedRCU/PERIOD(consumedRCU)",
      "Label": "Consumed RCUs",
      "ReturnData": false
    },
    {
      "Id": "utilizationPercentage",
      "Expression": "100*(m1/provisionedRCU)",
      "Label": "Utilization Percentage",
      "ReturnData": true
    }
  ],
  "StartTime": "<start-time>",
  "EndTime": "<end-time>",
  "ScanBy": "TimestampDescending",
  "MaxDatapoints": 24
}
```

Una vez que haya creado los archivos, podrá empezar a recuperar los datos de uso.

1. Para recuperar los datos de utilización de escritura, ejecute el siguiente comando:

   ```
   aws cloudwatch get-metric-data --cli-input-json file://write-calc.json
   ```

2. Para recuperar los datos de utilización de lectura, ejecute el siguiente comando:

   ```
   aws cloudwatch get-metric-data --cli-input-json file://read-calc.json
   ```

El resultado de ambas consultas será una serie de puntos de datos en formato JSON que se utilizarán para el análisis. El resultado dependerá de la cantidad de puntos de datos que haya

especificado, del periodo y de los propios datos de carga de trabajo específicos. Podría tener un aspecto similar al siguiente:


```
{
    "MetricDataResults": [
        {
            "Id": "utilizationPercentage",
            "Label": "Utilization Percentage",
            "Timestamps": [
                "2022-02-22T05:00:00+00:00",
                "2022-02-22T04:00:00+00:00",
                "2022-02-22T03:00:00+00:00",
                "2022-02-22T02:00:00+00:00",
                "2022-02-22T01:00:00+00:00",
                "2022-02-22T00:00:00+00:00",
                "2022-02-21T23:00:00+00:00"
            ],
            "Values": [
                91.55364583333333,
                55.066631944444445,
                2.6114930555555556,
                24.9496875,
                40.94725694444445,
                25.61819444444444,
                0.0
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

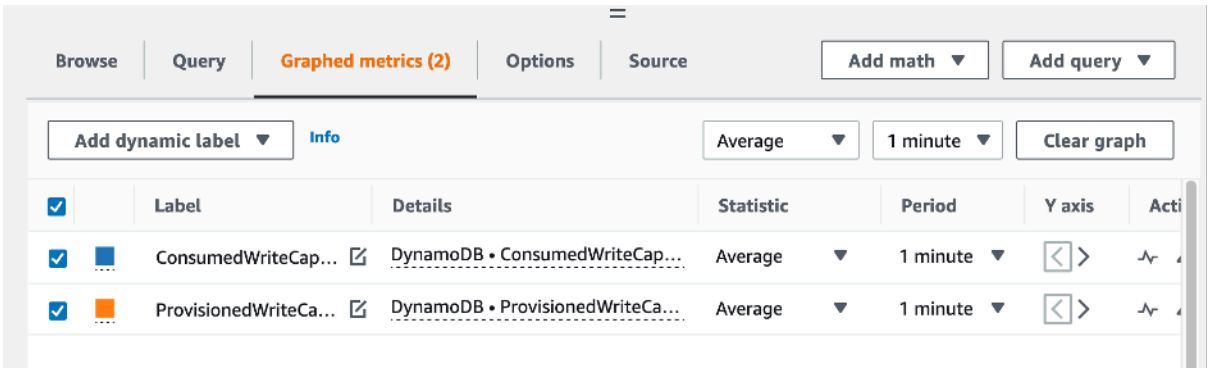

> **Note**
>
> Si especifica un periodo corto y un intervalo de tiempo largo, es posible que tenga que modificar `MaxDatapoints`, que es el valor predeterminado establecido en 24 en el script. Este representa un punto de datos por hora y 24 por día.

Consola de administración de AWS

1. Inicie sesión en la Consola de administración de AWS y navegue hasta la página del servicio de CloudWatch. Seleccione la Región de AWS apropiada si es necesario.
2. Localice la sección Métricas en la barra de navegación izquierda y seleccione Todas las métricas.
3. Esto abrirá un panel con dos paneles. El panel superior verá el gráfico y en el panel inferior tendrá las métricas que desea representar en un gráfico. Elija DynamoDB.
4. Elija la pestaña Métricas de tabla. Esto le mostrará las tablas en la región actual.
5. Utilice el cuadro de búsqueda para buscar el nombre de la tabla y elegir las métricas de la operación de escritura: `ConsumedWriteCapacityUnits` y `ProvisionedWriteCapacityUnits`

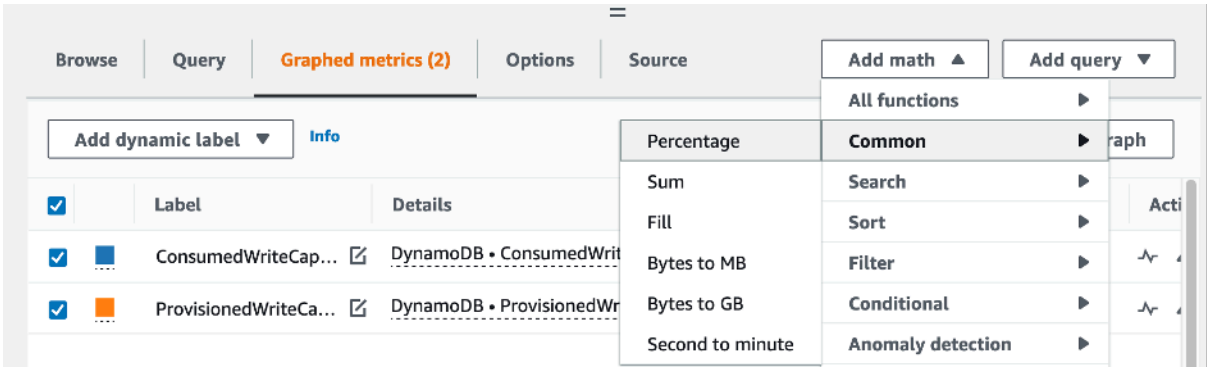
   > **Note**
   >
   > En este ejemplo se habla de las métricas de las operaciones de escritura, pero también puede utilizar estos pasos para representar gráficamente las métricas de las operaciones de lectura.

6. Seleccione la pestaña Métricas diagramadas (2) para modificar las fórmulas. De forma predeterminada, CloudWatch seleccionará la función estadística Media para los gráficos.

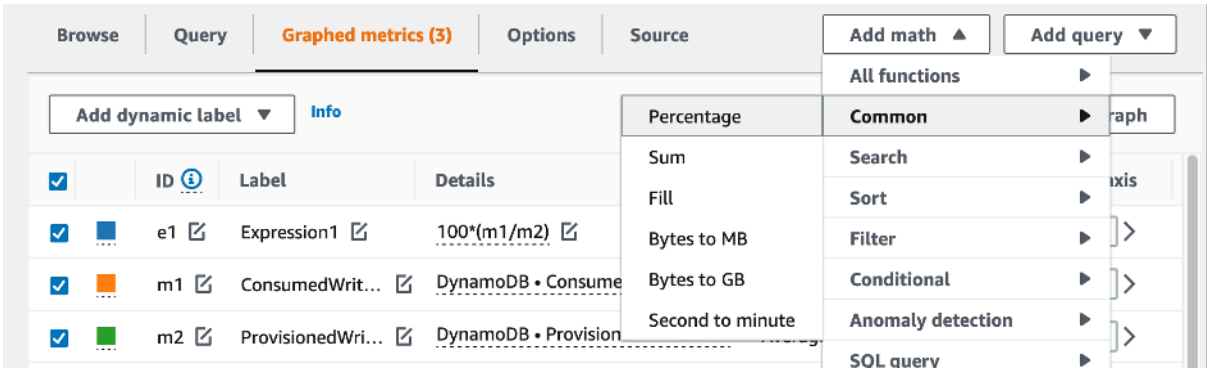


7. Con ambas métricas gráficas seleccionadas (la casilla de verificación de la izquierda), seleccione el menú Agregar matemática, seguido de Común y, a continuación, seleccione la función Porcentaje. Repita el procedimiento dos veces.

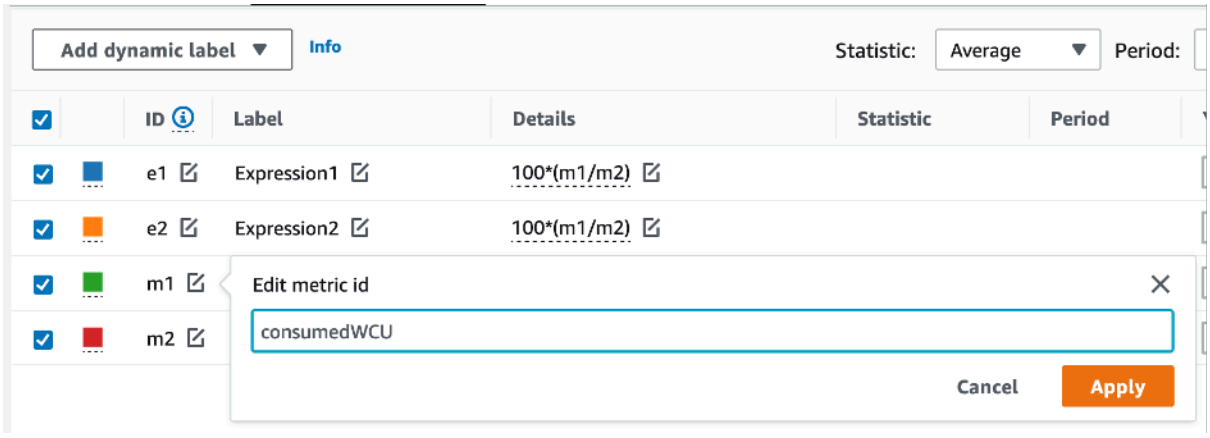
   Selección por primera vez de la función Percentage (Porcentaje):



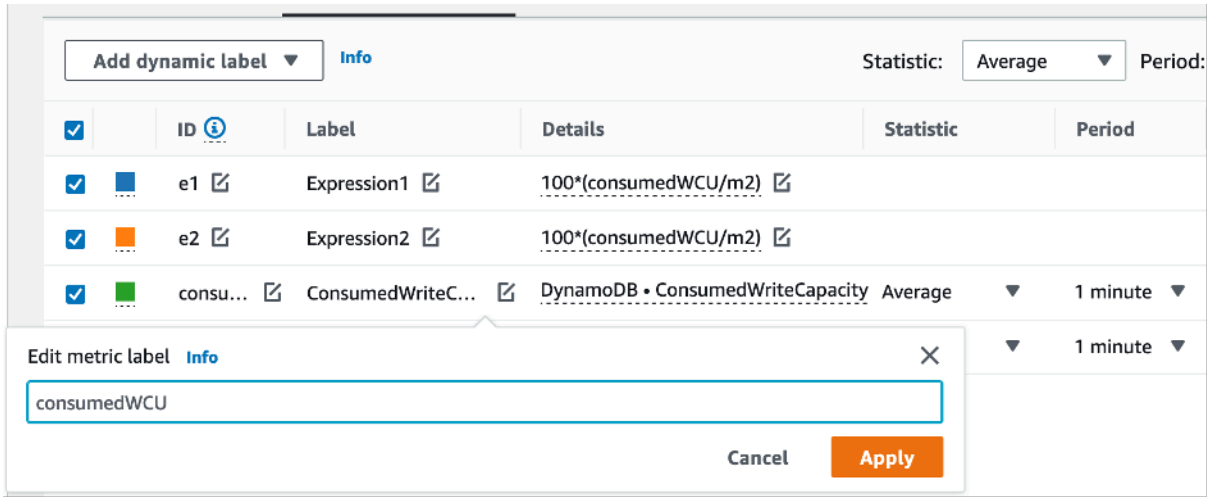
Selección por segunda vez de la función Percentage (Porcentaje):



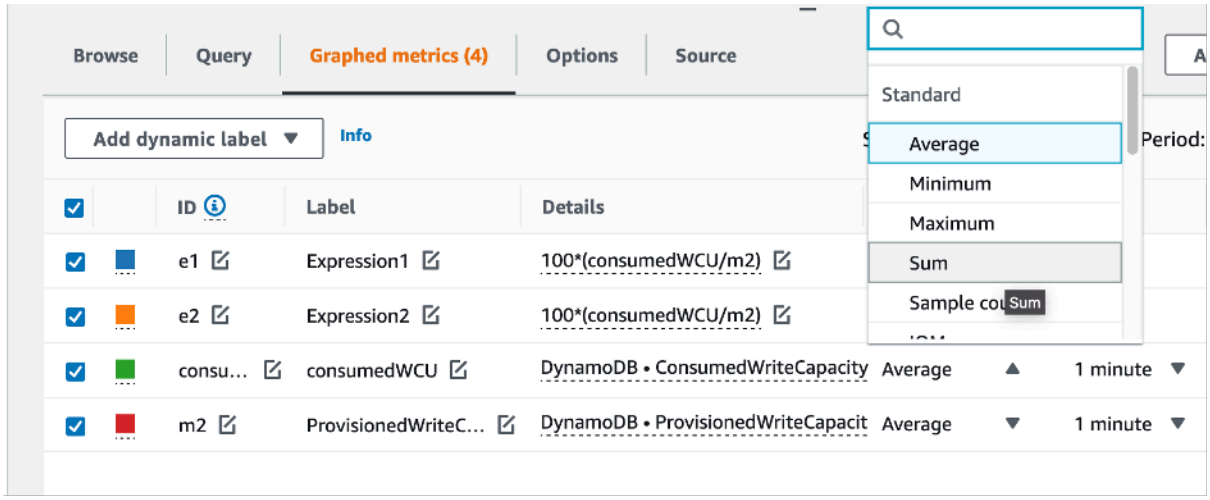
8. En este punto, debería tener cuatro métricas en el menú inferior. Vamos a trabajar en el cálculo de `ConsumedWriteCapacityUnits`. Para ser coherentes, necesitamos hacer coincidir los nombres con los que usamos en la sección de la AWS CLI. Haga clic en ID de m1 y cambie este valor a consumedWCU.

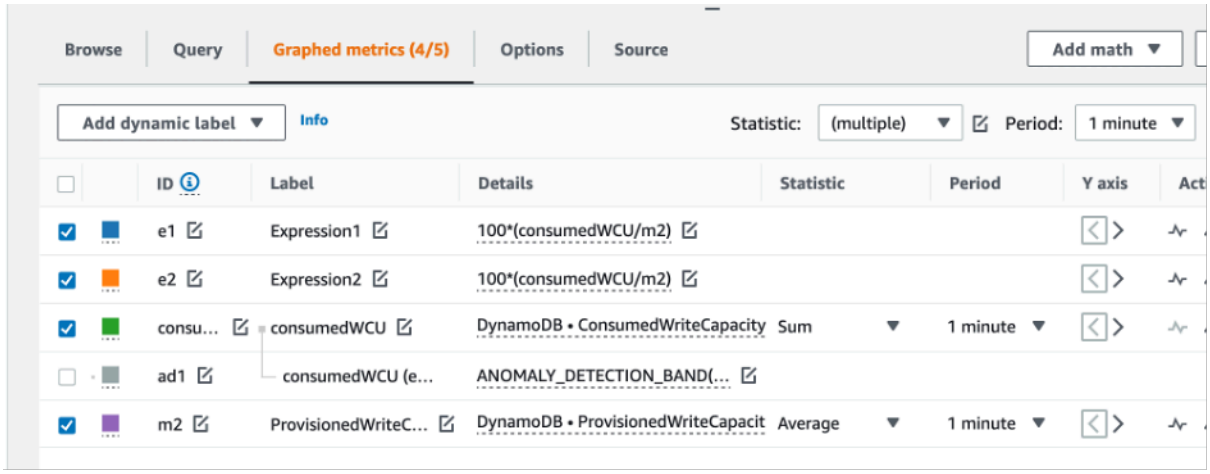


Cambie el nombre de la etiqueta ConsumedWriteCapacityUnit a **`consumedWCU`**.

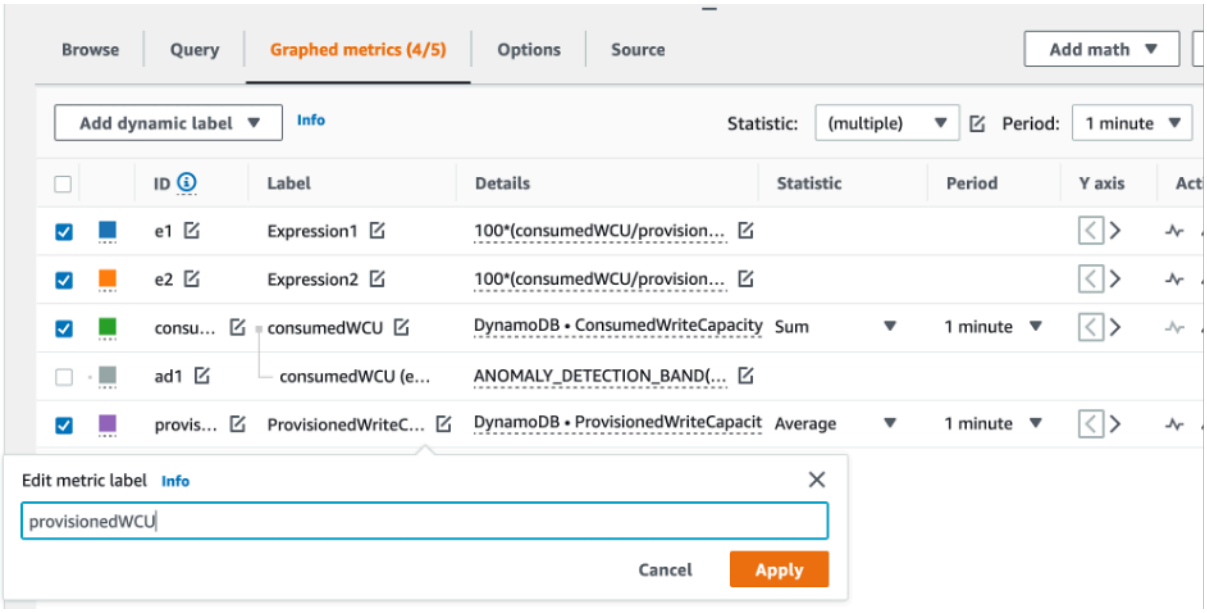


9. Cambie la estadística de Average (Media) a Sum (Suma). Esta acción creará automáticamente otra métrica llamada ANOMALY_DETECTION_BAND. Para conocer el alcance de este procedimiento, vamos a ignorarlo eliminando la casilla de verificación de la métrica ad1 recién generada.

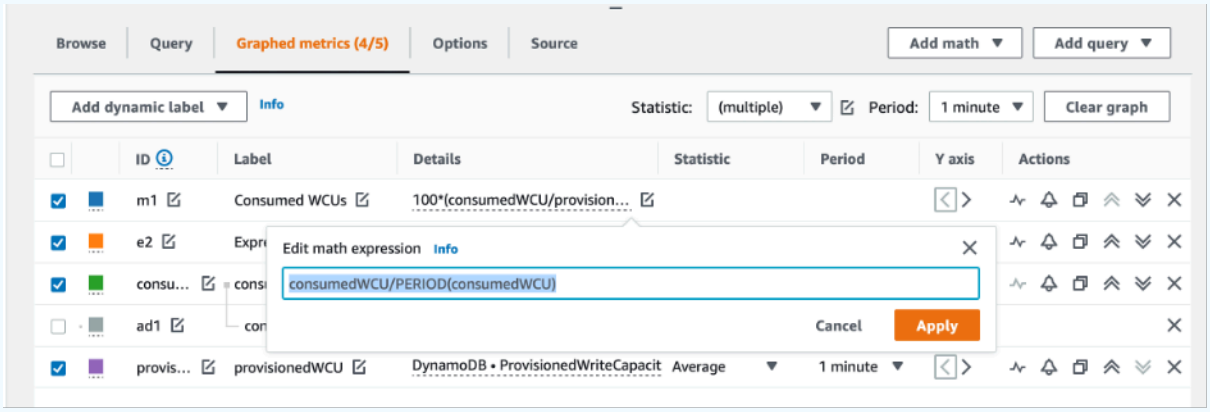




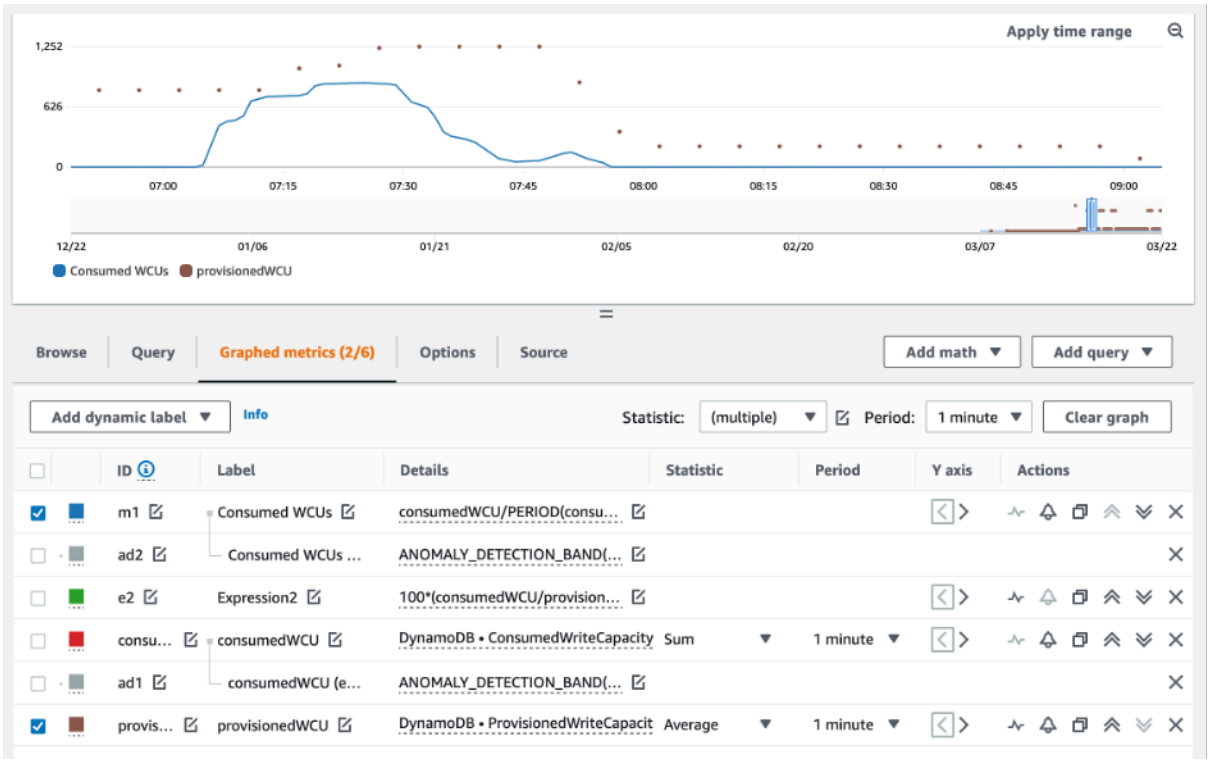
10. Repita el paso 8 para cambiar el nombre de ID de m2 a provisionedWCU. Deje la estadística establecida en Media.



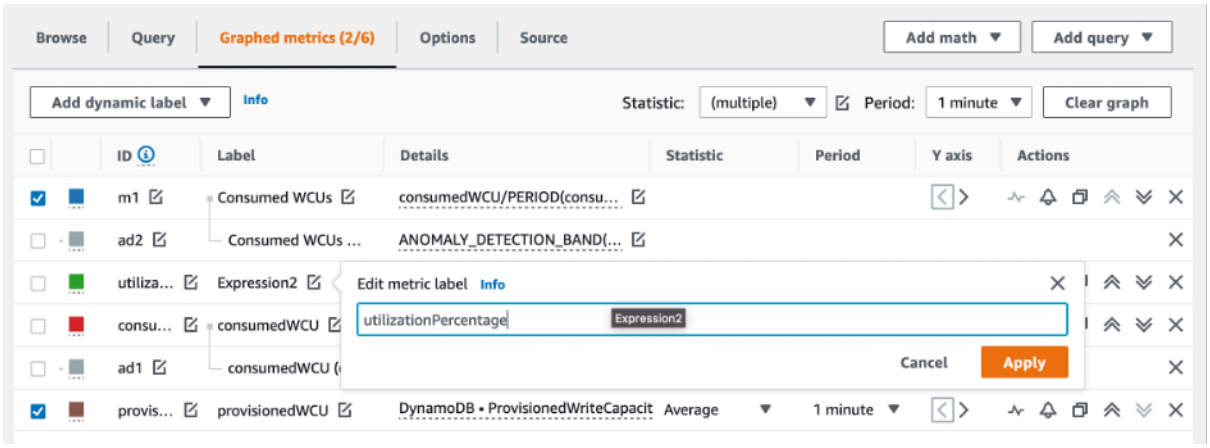
11. Seleccione la etiqueta Expression1 y actualice el valor a m1 y la etiqueta a Consumed WCUs (WCU consumidas).

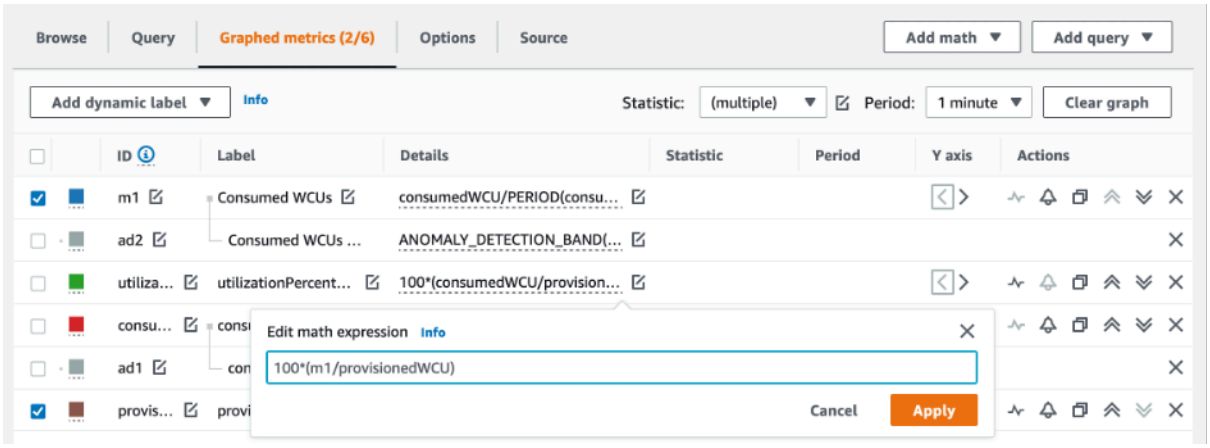
> **Note**
>
> Asegúrese de haber seleccionado solo m1 (casilla de verificación de la izquierda) y provisionedWCU para visualizar correctamente los datos. Actualice la fórmula haciendo clic en Details (Detalles) y cambiándola a consumedWCU/PERIOD(consumedWCU). Es posible que este paso también genere otra métrica de ANOMALY_DETECTION_BAND, pero para el alcance de este procedimiento podemos ignorarla.
>
> 
>

12. Ahora debería tener dos gráficos: uno que indica las WCU aprovisionadas en la tabla y otro que indica las WCU consumidas. Es posible que la forma del gráfico sea diferente a la de abajo, pero puede usarla como referencia:



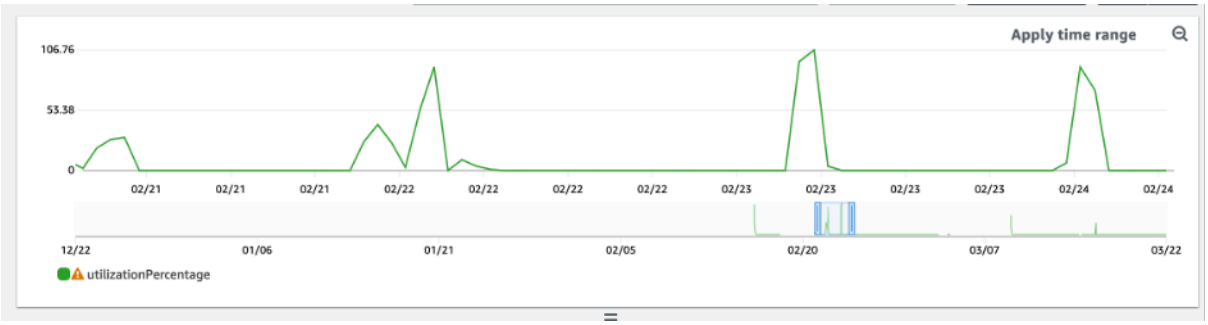
13. Actualice la fórmula porcentual seleccionando el gráfico Expression2 (e2). Cambie el nombre de las etiquetas e ID a utilizationPercentage. Cambie el nombre de la fórmula para que coincida con 100*(m1/provisionedWCU).





14. Elimine la casilla de verificación de todas las métricas, excepto de utilizationPercentage, para visualizar los patrones de utilización. El intervalo predeterminado está establecido en 1 minuto, pero no dude en modificarlo según lo necesite.

Esta es una vista de un periodo de tiempo más largo, así como un periodo mayor de 1 hora. Puede ver que hay algunos intervalos en los que la utilización fue superior al 100 %, pero esta carga de trabajo en particular tiene intervalos más largos sin ningún uso.



En este punto, es posible que obtenga resultados diferentes de las imágenes de este ejemplo. Todo depende de los datos de la carga de trabajo. Los intervalos con una utilización superior al 100 % tienden a limitar los eventos. DynamoDB ofrece capacidad de ampliación, pero tan pronto como se complete la capacidad de ampliación, se limitará todo lo que supere el 100 %.

Cómo identificar tablas de DynamoDB con falta de aprovisionamiento

Para la mayoría de las cargas de trabajo, una tabla se considera con falta de aprovisionamiento cuando consume de forma continua más del 80 % de la capacidad aprovisionada.

La capacidad de ampliación es una característica de DynamoDB que permite a los clientes consumir temporalmente más RCU o WCU de las aprovisionadas originalmente (más que el rendimiento aprovisionado por segundo que se definió en la tabla). La capacidad de ampliación se creó para absorber los aumentos repentinos del tráfico debido a eventos especiales o picos de uso. Esta capacidad de ampliación no dura para siempre. Tan pronto como se agoten las RCU y WCU no utilizadas, se verá limitado si intenta consumir más capacidad de la aprovisionada. Cuando el tráfico de la aplicación se acerca a la tasa de utilización del 80 %, el riesgo de limitación es significativamente mayor.

La regla de la tasa de utilización del 80 % varía según la estacionalidad de los datos y el crecimiento del tráfico. Considere los siguientes escenarios:

- Si el tráfico se ha mantenido estable a una tasa de utilización de aproximadamente el 90 % durante los últimos 12 meses, la tabla tiene la capacidad adecuada
- Si el tráfico de la aplicación crece a un ritmo del 8 % mensual en menos de 3 meses, llegará al 100 %
- Si el tráfico de la aplicación crece a un ritmo del 5 % mensual en un poco más de 4 meses, llegará al 100 %

Los resultados de las consultas anteriores proporcionan una imagen de la tasa de utilización. Úselos como guía para evaluar con más detalle otras métricas que pueden ayudarle a aumentar la capacidad de la tabla según sea necesario (por ejemplo, una tasa de crecimiento mensual o semanal). Trabaje con el equipo de operaciones para definir cuál es un buen porcentaje para la carga de trabajo y las tablas.

Hay escenarios especiales en los que los datos están sesgados cuando los analizamos a diario o semanalmente. Por ejemplo, en el caso de las aplicaciones estacionales que tienen picos de uso durante las horas de trabajo (pero que luego se reducen a casi cero fuera del horario laboral), podría resultar beneficioso programar el escalado automático en el que se especifiquen las horas del día (y los días de la semana) para aumentar la capacidad aprovisionada y cuándo reducirla. En lugar de optar por una mayor capacidad para cubrir las horas más ocupadas, también puede aprovechar las configuraciones de escalado automático de tablas de DynamoDB si la estacionalidad es menos pronunciada.

> **Note**
>
> Al crear una configuración de escalado automático de DynamoDB para la tabla base, recuerde incluir otra configuración para cualquier GSI que esté asociado a la tabla.

Cómo identificar tablas de DynamoDB con exceso de aprovisionamiento

Los resultados de la consulta obtenidos de los scripts anteriores proporcionan los puntos de datos necesarios para realizar algunos análisis iniciales. Si el conjunto de datos presenta valores de utilización inferiores al 20 % durante varios intervalos, es posible que la tabla tenga sobreaprovisionamiento. Para definir con más detalle si necesita reducir el número de WCU y RCU, debe revisitar las demás lecturas en los intervalos.

Cuando las tablas contienen varios intervalos de uso bajo, puede beneficiarse mucho del uso de políticas de escalado automático, ya sea programando el escalado automático o simplemente configurando las políticas de escalado automático predeterminadas para la tabla, que se basan en la utilización.

Si tiene una carga de trabajo con una relación de limitación entre baja utilización y alta (Max(ThrottleEvents)/Min(ThrottleEvents) en el intervalo), esto podría ocurrir cuando tenga una carga de trabajo muy elevada, en la que el tráfico aumente considerablemente durante algunos días (u horas), pero en general el tráfico es continuamente bajo. En estos escenarios, puede resultar beneficioso utilizar el escalado automático programado.

El marco AWS Well-Architected Framework ayuda a los arquitectos en la nube a crear infraestructuras seguras, de alto rendimiento, resilientes y eficientes para una gran variedad de aplicaciones y cargas de trabajo. Construido en torno a seis pilares (excelencia operativa, seguridad, fiabilidad, eficiencia del rendimiento, optimización de costes y sostenibilidad) AWS Well-Architected proporciona un enfoque coherente para que clientes y socios evalúen arquitecturas e implementen diseños escalables.

Los enfoques de AWS Well-Architected amplían la orientación que ofrece AWS Well-Architected a determinados dominios sectoriales y tecnológicos. El enfoque Well-Architected de Amazon DynamoDB se centra en las cargas de trabajo de DynamoDB. Proporciona prácticas recomendadas, principios de diseño y preguntas para evaluar y revisar una carga de trabajo de DynamoDB. La realización de una revisión de enfoque Well-Architected de Amazon DynamoDB le proporcionará formación y orientación sobre los principios de diseño recomendados en relación con cada uno de

los pilares el enfoque de AWS Well-Architected. Esta orientación se basa en nuestra experiencia de trabajo con clientes de diversos sectores, segmentos, tamaños y zonas geográficas.

Como resultado directo de la revisión del enfoque Well-Architected, recibirá un resumen de recomendaciones procesables para optimizar y mejorar su carga de trabajo de DynamoDB.

Realización de la revisión del enfoque Well-Architected de Amazon DynamoDB

La revisión del enfoque Well-Architected de DynamoDB normalmente la realiza un arquitecto de soluciones de AWS junto con el cliente, pero este también puede realizarla como autoservicio. Aunque recomendamos revisar los seis pilares Well-Architected como parte del enfoque Well-Architected de Amazon DynamoDB, también puede decidir priorizar su enfoque en uno o más pilares primero.

Encontrará información adicional e instrucciones para realizar una revisión del enfoque Well-Architected de Amazon DynamoDB en este vídeo y en la página de GitHub del enfoque Well-Architected de DynamoDB.

Los pilares del enfoque Well-Architected de Amazon DynamoDB

El enfoque Well-Architected de Amazon DynamoDB se basa en seis pilares:

Pilar de eficiencia de rendimiento

El pilar de eficiencia del rendimiento incluye la capacidad para utilizar los recursos de computación de forma eficaz a fin de que satisfagan los requisitos del sistema y para mantener dicha eficacia a medida que la demanda cambia y las tecnologías evolucionan.

Los principales principios de diseño de DynamoDB para este pilar giran en torno al modelado de los datos, la elección de claves de partición y claves de clasificación y la definición de índices secundarios basados en los patrones de acceso de la aplicación. Otras consideraciones son elegir el modo de rendimiento óptimo para la carga de trabajo, el ajuste del SDK de AWS y, cuando proceda, utilizar una estrategia óptima de almacenamiento en caché. Para obtener más información sobre estos principios de diseño, vea este vídeo detallado sobre el pilar de eficiencia de rendimiento del enfoque Well-Architected de DynamoDB.

Pilar de optimización de costos

El pilar de optimización de costes se centra en evitar costos innecesarios.

Los temas clave incluyen la comprensión y el control de dónde se gasta el dinero, la selección del número más apropiado y correcto de tipos de recursos, el análisis del gasto a lo largo del tiempo, el diseño de sus modelos de datos a fin de optimizar el costo para los patrones de acceso específicos de la aplicación y la ampliación para satisfacer las necesidades empresariales sin gastar en exceso.

Los principios clave del diseño de optimización de costos para DynamoDB giran en torno a elegir el modo de capacidad y la clase de tabla más adecuados para sus tablas y evitar el sobreaprovisionamiento de capacidad, ya sea mediante el modo de capacidad bajo demanda, o el modo de capacidad aprovisionada con escalado automático. Entre las consideraciones adicionales se incluyen un modelado de datos y una consulta eficientes para reducir la cantidad de capacidad consumida, reservar partes de la capacidad consumida a precio de descuento, minimizar el tamaño de los elementos, identificar y eliminar los recursos no utilizados y utilizar TTL para eliminar automáticamente los datos caducados sin costo alguno. Para obtener más información sobre estos principios de diseño, vea este vídeo detallado sobre el pilar de eficiencia de optimización de costos del enfoque Well-Architected de DynamoDB.

Consulte Optimización de costos para obtener información adicional sobre las prácticas recomendadas de optimización de costos para DynamoDB.

Pilar de excelencia operativa

El pilar de la excelencia operativa se centra en el funcionamiento y la supervisión de los sistemas para ofrecer valor empresarial, y en la mejora continua de los procesos y los procedimientos. Los temas clave incluyen la automatización de los cambios, la respuesta a los eventos y la definición de estándares para administrar las operaciones diarias.

Los principales principios de diseño de excelencia operativa para DynamoDB incluyen supervisar las métricas de DynamoDB a través de Amazon CloudWatch y AWS Config, además de alertar y corregir automáticamente cuando se superan los umbrales predefinidos o se detectan reglas que no se cumplen. Otras consideraciones adicionales son la definición de los recursos de DynamoDB mediante la infraestructura como código y el aprovechamiento de las etiquetas para una mejor organización, identificación y contabilidad de costos de sus recursos de DynamoDB. Para obtener más información sobre estos principios de diseño, vea este vídeo detallado sobre el pilar de excelencia operativa del enfoque Well-Architected de DynamoDB.

Pilar de fiabilidad

El pilar de fiabilidad se centra en garantizar que una carga de trabajo desempeñe la función prevista de manera correcta y coherente cuando se espera que lo haga. Una carga de trabajo resiliente se recupera rápidamente de los errores para satisfacer la demanda empresarial y de los clientes. Entre

los temas clave se incluyen el diseño de sistemas distribuidos, la planificación de la recuperación y cómo gestionar el cambio.

Los principios de diseño de fiabilidad esenciales para DynamoDB giran en torno a elegir la estrategia de copia de seguridad y retención en función de sus requisitos de RPO y RTO, utilizar tablas globales de DynamoDB para cargas de trabajo multirregionales o escenarios de recuperación de desastres entre regiones con un RTO bajo, implementar la lógica de reintento con retroceso exponencial en la aplicación mediante la configuración y el uso de estas capacidades en el SDK de AWS, y supervisar las métricas de DynamoDB a través de Amazon CloudWatch y alertar y corregir automáticamente cuando se superen los umbrales predefinidos. Para obtener más información sobre estos principios de diseño, vea este vídeo detallado sobre el pilar de fiabilidad del enfoque Well-Architected de DynamoDB.

Pilar de seguridad

El pilar de seguridad se centra en proteger la información y los sistemas. Los temas clave incluyen la confidencialidad e integridad de los datos, la identificación y administración de quién puede hacer qué con la administración de privilegios, la protección de los sistemas y el establecimiento de controles para detectar eventos de seguridad.

Los principales principios de diseño de seguridad para DynamoDB son cifrar los datos en tránsito con HTTPS, elegir el tipo de claves para el cifrado de datos en reposo y definir los roles de IAM y las políticas para autenticar, autorizar y proporcionar acceso detallado a los recursos de DynamoDB. Las consideraciones adicionales incluyen la auditoría del plano de control de DynamoDB y las operaciones del plano de datos a través de AWS CloudTrail. Para obtener más información sobre estos principios de diseño, vea este vídeo detallado sobre el pilar de seguridad del enfoque Well-Architected de DynamoDB.

Consulte Seguridad para obtener información adicional sobre la seguridad para DynamoDB.

Pilar de sostenibilidad

El pilar de la sostenibilidad se centra en minimizar el impacto medioambiental de la ejecución de cargas de trabajo en la nube. Los temas clave incluyen un modelo de responsabilidad compartida para la sostenibilidad, la comprensión del impacto y la maximización de la utilización para minimizar los recursos necesarios y reducir los impactos posteriores.

Los principales principios de diseño sostenible para DynamoDB incluyen la identificación y eliminación de los recursos de DynamoDB no utilizados, lo que evita el aprovisionamiento excesivo mediante el uso del modo de capacidad bajo demanda o el modo de capacidad aprovisionada con

escalado automático, la realización de consultas eficientes para reducir la cantidad de capacidad que se consume y la reducción de la huella de almacenamiento mediante la compresión de datos y la eliminación de datos antiguos mediante el uso de TTL. Para obtener más información sobre estos principios de diseño, vea este vídeo detallado sobre el pilar de sostenibilidad del enfoque Well-Architected de DynamoDB.

Prácticas recomendadas para diseñar claves de partición y utilizarlas con eficacia en DynamoDB

La clave principal que identifica de manera inequívoca cada elemento de una tabla de Amazon DynamoDB puede ser simple (ser únicamente una clave de partición) o compleja (estar formada por una clave de partición y una clave de ordenación).

La aplicación se debería diseñar con la idea de que la actividad será uniforme en todas las claves de las particiones de la tabla y sus índices secundarios. Puede determinar los patrones de acceso que requiere su aplicación y las unidades de lectura y escritura que requiere cada tabla e índice secundario.

> **Note**
>
> La capacidad de adaptación se aplica al modo bajo demanda y a la capacidad aprovisionada.

Cada partición de una tabla de DynamoDB está diseñada para ofrecer una capacidad máxima de 3000 unidades de lectura por segundo y 1000 unidades de escritura por segundo. Una unidad de lectura representa una lectura altamente coherente por segundo o dos lecturas coherentes posteriores por segundo, para elementos con un tamaño máximo de 4 KB. Una unidad de escritura representa una operación de escritura por segundo para un elemento con un tamaño máximo de 1 KB.

Debe tener en cuenta el tamaño del elemento al evaluar los límites de rendimiento de las particiones de la tabla. Por ejemplo, si la tabla tiene un tamaño de elemento de 20 KB, una sola operación de lectura coherente consumirá 5 unidades de lectura. Esto significa que puede realizar simultáneamente 600 operaciones de lectura coherente por segundo en ese único elemento antes de alcanzar los límites de partición. El rendimiento total de todas las particiones de la tabla puede estar limitado por el rendimiento aprovisionado en el modo aprovisionado o por el límite de rendimiento a nivel de tabla en el modo bajo demanda. Para obtener más información, consulte Service Quotas.

Temas

- Diseño de claves de partición para distribuir la carga de trabajo en DynamoDB
- Uso de la partición de escritura para distribuir las cargas de trabajo uniformemente en la tabla de DynamoDB
- Distribución de la actividad de escritura de forma eficiente al cargar los datos en DynamoDB

Diseño de claves de partición para distribuir la carga de trabajo en DynamoDB

La parte formada por la clave de partición de la clave primaria de una tabla determina las particiones lógicas en las que se almacenan los datos de la tabla. Esto, a su vez, afecta a las particiones físicas subyacentes. Un diseño de clave de partición que no distribuya las solicitudes de E/S de un modo eficaz podría crear particiones “activas” que provoquen una limitación controlada y no utilicen la capacidad de E/S aprovisionada de forma eficaz.

El uso óptimo del diseño aprovisionado de una tabla no solo depende de los patrones de carga de trabajo de los elementos individuales, sino también del diseño de la clave de partición. Esto no significa que haya que obtener acceso a todos los valores de la clave de partición para lograr un nivel de rendimiento eficaz ni que el porcentaje de valores de la clave de partición a los que se obtiene acceso tenga que ser elevado. Lo que significa es que a cuantos más valores diferentes de la clave de partición tenga acceso la carga de trabajo, más se distribuirán estas solicitudes entre el espacio particionado. En general, el rendimiento aprovisionado se utilizará con más eficacia cuanto mayor sea la proporción de los valores de la clave de partición a los que se obtiene acceso en comparación con el número total de valores de la clave de partición.

A continuación, se muestra una comparación de la eficiencia del rendimiento aprovisionado de algunos esquemas de claves de partición comunes.

Valor de clave de partición	Uniformidad
ID de usuario, en caso de que la aplicación tenga muchos usuarios.	Buena
Código de estado, aunque solo hay algunos códigos de estado posibles.	Mala

Valor de clave de partición	Uniformidad
Fecha de creación del elemento, redondeada al periodo más próximo (por ejemplo, día, hora o minuto).	Mala
ID de dispositivo, en un caso en que cada dispositivo obtiene acceso a los datos a intervalos relativamente similares.	Buena
ID del dispositivo, donde, aunque se hace un seguimiento de muchos dispositivos, uno de ellos es muchísimo más popular que los demás.	Mala

Si una tabla solo contiene un número reducido de valores de clave de partición, puede ser conveniente distribuir las operaciones de escritura entre más valores de clave de partición distintos. En otras palabras, conviene estructurar los elementos de clave principal de tal forma que se evite un valor de clave de partición "caliente" (muy solicitado) que ralentice el desempeño general.

Por ejemplo, tomemos una tabla con una clave principal compuesta. La clave de partición representa la fecha de creación del elemento, redondeada al día más próximo. La clave de ordenación es un identificador de elemento. En un día determinado (por ejemplo, `2014-07-09`), todos los elementos nuevos se escriben en ese mismo valor de clave de partición (y en la partición física correspondiente).

Si la tabla completa cabe en una sola partición (teniendo en cuenta el crecimiento que registrarán los datos con el paso del tiempo) y los requisitos de rendimiento de lectura y escritura de la aplicación no rebasan la capacidad de lectura y escritura de una sola partición, la aplicación no debería verse afectada por una limitación controlada inesperada como resultado de este particionamiento.

Para utilizar NoSQL Workbench para DynamoDB como ayuda para visualizar su diseño de clave de partición, consulte Creación de modelos de datos con NoSQL Workbench.

Uso de la partición de escritura para distribuir las cargas de trabajo uniformemente en la tabla de DynamoDB

Una forma de distribuir mejor las escrituras entre un espacio de claves de particiones en Amazon DynamoDB consiste en ampliar el espacio. Esto puede hacerse de diferentes maneras. Puede agregar un número aleatorio a los valores de clave de partición para distribuir los elementos entre particiones. O puede usar un número que se calcula en función de algo que esté consultando.

Partición con sufijos aleatorios

Una estrategia para distribuir las cargas de forma más uniforme en un espacio de claves de partición consiste en añadir un número aleatorio al final de los valores de la clave de partición. De ese modo, las escrituras se distribuyen aleatoriamente por un espacio mayor.

Por ejemplo, para una clave de partición que representara la fecha de hoy, podría elegir un número aleatorio comprendido entre `1` y `200` y concatenarlo a la fecha como sufijo. De este modo, se generarían valores de clave de partición, como `2014-07-09.1`, `2014-07-09.2` y así sucesivamente hasta `2014-07-09.200`. Al aplicar un número aleatorio a la clave de partición, las escrituras que se producen en la tabla de cada día se distribuyen uniformemente por varias particiones. Como resultado, se mejora el paralelismo y el rendimiento general.

Sin embargo, si quisiera leer todos los elementos de un día determinado, tendría que hacer una consulta de todos los sufijos y fusionar después los resultados. Por ejemplo, primero debería emitir una solicitud `Query` para el valor de la clave de partición `2014-07-09.1`. A continuación, emita otra `Query` para `2014-07-09.2` y, así sucesivamente, hasta `2014-07-09.200`. Por último, la aplicación tendría que fusionar los resultados de todas estas solicitudes `Query`.

Partición con sufijos calculados

Aplicar una estrategia de aleatorización puede mejorar considerablemente el rendimiento de la escritura. Sin embargo, dificulta la lectura de un elemento concreto, ya que no es posible saber qué sufijo se utilizó al escribir el elemento. Para facilitar la lectura de elementos concretos, puede usar una estrategia diferente. En lugar de utilizar un número aleatorio para distribuir los elementos entre las particiones, puede utilizar un número que se calculará en función de algo que se quiere consultar.

Consideremos el ejemplo anterior, en el que una tabla utiliza la fecha de hoy en la clave de partición. Ahora, supongamos que cada elemento tiene un atributo `OrderId` (ID de pedido) accesible y que lo que suele necesitar con más frecuencia es buscar elementos por el ID de pedido además de por

la fecha. Antes de que la aplicación escriba el elemento en la tabla, se podría calcular un sufijo hash basado en el ID de pedido y anexarlo a la fecha de la clave de partición. El cálculo podría generar un número comprendido entre 1 y 200 que esté bastante bien distribuido, igual que el que se genera con la estrategia aleatoria.

Probablemente, bastaría con realizar un sencillo cálculo, como el producto de los valores de punto de código UTF-8 de los caracteres del ID de pedido, módulo 200, +1. El valor de la clave de partición sería la fecha junto con el resultado del cálculo.

Con esta estrategia, las escrituras se distribuyen de manera uniforme entre los valores de clave de partición y, por lo tanto, entre las particiones físicas. Es fácil realizar una operación `GetItem` en un elemento y una fecha determinados, ya que se puede calcular el valor de clave de partición de un valor de `OrderId` concreto.

Para leer todos los elementos de un día determinado, también en este caso deberá utilizar una solicitud `Query` para cada una de las claves `2014-07-09.N` (donde `N` es un valor comprendido entre 1 y 200) y la aplicación tendría que fusionar todos los resultados. El beneficio es que evitaría que un único valor de clave de partición caliente acaparase toda la carga de trabajo.

> **Note**
>
> Para obtener información sobre una estrategia más eficaz diseñada específicamente para administrar datos de serie temporal de gran volumen, consulte Datos de serie temporal.

Distribución de la actividad de escritura de forma eficiente al cargar los datos en DynamoDB

Normalmente, cuando carga datos de otros orígenes de datos, Amazon DynamoDB particiona los datos de la tabla en varios servidores. Obtendrá un rendimiento mejor si carga los datos en todos los servidores asignados al mismo tiempo.

Por ejemplo, suponga que desea cargar mensajes de usuarios en una tabla de DynamoDB que utilice una clave principal compuesta con `UserID` como clave de partición y `MessageID` como clave de clasificación.

Cuando cargue los datos, un enfoque que puede tomar es cargar todos los elementos de mensaje para cada usuario, un usuario tras otro:

UserID	MessageID
U1	1
U1	2
U1	...
U1	... hasta 100
U2	1
U2	2
U2	...
U2	... hasta 200

El problema en este caso es que no está distribuyendo las solicitudes de escritura a DynamoDB entre los valores de clave de partición. Toma un valor de clave de partición a la vez y carga todos sus elementos, antes de pasar al siguiente valor de clave de partición y hacer lo mismo.

En segundo plano, DynamoDB particiona los datos de las tablas entre varios servidores. Para utilizar toda la capacidad de rendimiento que se ha aprovisionado para la tabla, debe distribuir la carga de trabajo entre los valores de clave de partición. Al dirigir una cantidad desigual de carga a elementos que tienen el mismo valor de clave de partición, es posible que no pueda utilizar plenamente los recursos que DynamoDB ha aprovisionado para la tabla.

Puede distribuir el trabajo de carga utilizando la clave de ordenación para cargar un elemento de cada valor de clave de partición, luego otro elemento de cada valor de clave de partición y así sucesivamente:

UserID	MessageID
U1	1
U2	1
U3	1

UserID	MessageID
...	...
U1	2
U2	2
U3	2
...	...

Cada carga que se lleva a cabo en esta secuencia utiliza un valor de clave de partición distinto, por lo que más servidores de DynamoDB están ocupados simultáneamente mejorando así el rendimiento.

Prácticas recomendadas sobre el uso de claves de clasificación para organizar datos en DynamoDB

En una tabla de Amazon DynamoDB, la clave principal que identifica de manera inequívoca cada elemento de la tabla puede constar de una clave de partición y una clave de clasificación.

Si las claves de ordenación están bien diseñadas, tienen dos beneficios principales:

- Recopilan información relacionada en un único lugar en el que los datos se pueden consultar de forma eficaz. Si se diseñan meticulosamente, permiten recuperar grupos de elementos relacionados que suelen ser necesarios utilizando consultas de rango con operadores, como `begins_with`, `between`, >, <, etc.
- Las claves de ordenación compuestas permiten definir en los datos relaciones jerárquicas (de uno a varios) que pueden consultarse en cualquier nivel de la jerarquía.

  Por ejemplo, en una tabla con ubicaciones geográficas, la clave de ordenación podría estructurarse del modo siguiente:

  ```
  [country]#[region]#[state]#[county]#[city]#[neighborhood]
  ```

  Esto le permitiría crear consultas de rango eficaces para una lista de ubicaciones en cualquiera de estos niveles de agregación, desde `country` hasta `neighborhood`, pasando por cualquier elemento intermedio.

Uso de claves de clasificación para el control de versiones

Muchas aplicaciones deben conservar el historial de revisiones de los elementos por motivos de auditoría o conformidad y para poder recuperar fácilmente la versión más reciente. Existe un patrón de diseño eficaz que permite hacerlo mediante prefijos de clave de ordenación:

- Cree dos copias de cada elemento nuevo: una copia deber tener el prefijo de número de versión 0 (por ejemplo, `v0_`) al principio de la clave de ordenación mientras que la otra debe tener el prefijo de número de versión 1 (por ejemplo, `v1_`).
- Cada vez que el elemento se actualice, utilice el siguiente prefijo de versión en la clave de ordenación de la versión actualizada y copie el contenido actualizado en el elemento cuyo prefijo de versión es 0. Esto significa que la última versión de cada elemento puede localizarse fácilmente a través del prefijo 0.

Por ejemplo, un fabricante de piezas podría utilizar un esquema como el de la siguiente ilustración.

Primary Key		Data-Item Attributes...				
Partition Key *Equipment_ID*	Sort Key *(varies)*	Attribute 1	Attribute 2	Attribute 3	Attribute 4	...
Equipment_1	Details	Name: Biphasic Cardiometer *(equipment name)*	Factory_ID: S14_Tukwila *(factory where manufactured)*	Line_ID R_7 *(assembly-line ID)*		
	v0_Audit	Auditor: Padma *(name of the auditor)*	Latest: 3 *(most recent audit version)*	Time: 2018-04-15T11:00 *(audit date and time)*	Result Passed *(audit result)*	...etc.
	v1_Audit	Auditor: Rick *(name of the auditor)*	Time: 2018-03-14T11:00 *(audit date and time)*	Result Open *(audit result)*	Report: 0943922EKG14 *(detailed problem report in S3)*	...etc.
	v2_Audit	Auditor: George *(name of the auditor)*	Time: 2018-03-18T11:00 *(audit date and time)*	Result Open *(audit result)*	Report: 0943923EKG15 *(detailed problem report in S3)*	...etc.
	v3_Audit	Auditor: Padma *(name of the auditor)*	Time: 2018-04-15T11:00 *(audit date and time)*	Result Passed *(audit result)*	Report: x792 *(pass confirmation report)*	...etc.

Varios auditores realizan una serie de auditorías en el elemento `Equipment_1`. Los resultados de cada nueva auditoría se capturan en un nuevo elemento de la tabla, comenzando por el número de versión 1 e incrementando el número con cada revisión.

Cuando se agrega una nueva revisión, la capa de la aplicación sustituye el contenido del elemento que tiene la versión 0 (donde la clave de ordenación es igual a `v0_Audit`) por el contenido de la nueva versión.

Siempre que la aplicación tenga que recuperar el estado de auditoría más reciente, podrá consultar el prefijo de la clave de ordenación `v0_`.

Si la aplicación tiene que recuperar todo el historial de revisiones, podrá consultar todos los elementos situados bajo la clave de partición del elemento y filtrar el elemento `v0_`.

Este diseño también podría funcionar para realizar auditorías de varias piezas de un componente del equipo si se incluyera el ID de cada pieza en la clave de ordenación, detrás del prefijo.

Prácticas recomendadas para utilizar índices secundarios en DynamoDB

Con frecuencia, los índices secundarios resultan esenciales para admitir los patrones de consulta que necesita la aplicación. Sin embargo, un uso excesivo o inadecuado de estos índices podría incrementar los costos y reducir el rendimiento innecesariamente.

Contenido

- Directrices generales sobre los índices secundarios de DynamoDB
  - Uso eficaz de los índices
  - Elección cuidadosa de las proyecciones
  - Optimización de las consultas frecuentes para evitar recuperaciones
  - Tener en cuenta los límites de tamaño de la colección de elementos al crear índices secundarios locales
- Sacar partido de los índices dispersos
  - Ejemplos de índices dispersos en DynamoDB
- Uso de índices secundarios globales para consultas de agregación materializadas en DynamoDB
- Sobrecarga de índices secundarios globales en DynamoDB
- Uso de la partición de escritura del índice secundario global para las consultas de tabla selectivas en DynamoDB
  - Diseño de patrón
  - Estrategia de particiones
  - Consulta del GSI particionado
  - Consideraciones sobre la ejecución de consultas paralelas
  - Ejemplo de código
- Uso de índices secundarios globales para crear una réplica eventualmente consistente en DynamoDB

Directrices generales sobre los índices secundarios de DynamoDB

Amazon DynamoDB admite dos tipos de índices secundarios:

- Índice secundario global (GSI): índice con una clave de partición y una clave de clasificación que pueden ser diferentes de las de la tabla base. Un índice secundario global se considera "global" porque las consultas que se realizan en el índice pueden abarcar todos los datos de la tabla base y todas las particiones. Los índices secundarios globales no tienen limitaciones de tamaño y cuentan con su propia configuración de rendimiento aprovisionada para la actividad de lectura y escritura, que es diferente de la configuración de la tabla base.
- Índice secundario local (LSI): un índice que tiene la misma clave de partición que la tabla base, pero una clave de clasificación diferente. Un índice secundario local se considera "local" en el sentido de que el ámbito de todas sus particiones se corresponde con una partición de la tabla base que tiene el mismo valor de clave de partición. Por tanto, el tamaño total de los elementos indexados de cualquier valor de clave de partición no podrá ser superior a 10 GB. Además, los índices secundarios locales comparten la configuración de rendimiento aprovisionado para la actividad de lectura y lectura con la tabla que están indexando.

Cada tabla de DynamoDB puede tener hasta 20 índices secundarios globales (cuota predeterminada) y 5 índices secundarios locales.

Los índices secundarios globales suelen ser más útiles que los locales. Determinar qué tipo de índice utilizar dependerá también de los requisitos de su aplicación. Consulte the section called “Uso de índices” para comparar los índices secundarios globales y los índices secundarios locales, además de para obtener más información sobre cómo elegir entre ellos.

A continuación, se muestran algunos principios generales y patrones de diseño que deben tenerse en cuenta al crear índices en DynamoDB:

Temas

- Uso eficaz de los índices
- Elección cuidadosa de las proyecciones
- Optimización de las consultas frecuentes para evitar recuperaciones
- Tener en cuenta los límites de tamaño de la colección de elementos al crear índices secundarios locales

Uso eficaz de los índices

Mantenga el menor número de índices posible. No cree índices secundarios en atributos que no consulte con frecuencia. Los índices que se utilizan pocas veces provocan un aumento del almacenamiento y de los costos de E/S y no mejoran el rendimiento de la aplicación.

Elección cuidadosa de las proyecciones

Como los índices secundarios consumen almacenamiento y rendimiento aprovisionado, es importante que su tamaño sea lo menor posible. Además, cuanto menor sea el índice, mayor será el beneficio en términos de rendimiento en comparación con una consulta que abarque toda la tabla. Si las consultas suelen devolver tan solo un reducido subconjunto de atributos y el tamaño total de estos atributos es mucho menor que la totalidad del elemento, proyecte solamente aquellos atributos que solicite habitualmente.

Si prevé que la actividad de escritura de una tabla va a ser muy superior a la de lectura, siga estas prácticas recomendadas:

- Considere la posibilidad de proyectar menos atributos para minimizar el tamaño de los elementos que se escriben en el índice. No obstante, esto solo es aplicable si el tamaño de los atributos proyectados fuera mayor que una unidad de capacidad de escritura (1 KB). Por ejemplo, si el tamaño de una entrada de índice es de tan solo 200 bytes, DynamoDB la redondeará a 1 KB. Es decir, siempre y cuando los elementos del índice son pequeños, puede proyectar más atributos sin costo adicional.
- Evite proyectar atributos si sabe que apenas los va a necesitar en las consultas. Cada vez que se actualiza un atributo que está proyectado en un índice, también se actualiza el índice, lo que tiene un costo adicional. Los atributos no proyectados se pueden recuperar en un solicitud `Query`, aunque con un costo superior en lo que se refiere al rendimiento aprovisionado. Sin embargo, el costo de la consulta puede ser mucho más bajo que el de actualizar el índice con frecuencia.
- Especifique `ALL` solamente si desea que las consultas devuelvan el elemento de la tabla completo ordenado mediante una clave de ordenación distinta. Al proyectar todos los atributos, ya no será necesario recuperar la tabla. Sin embargo, en la mayoría de los casos, duplicará los costos de almacenamiento y de actividad de escritura.

Sopese la necesidad de mantener los índices con el menor tamaño posible frente a la necesidad de minimizar las actualizaciones lo máximo posible, tal y como se explica en la sección siguiente.

Optimización de las consultas frecuentes para evitar recuperaciones

Para acelerar al máximo las consultas con la mínima latencia posible, proyecte todos los atributos que crea que se van a devolver en esas consultas. En particular, si consulta un índice secundario local para buscar atributos que no están proyectados, DynamoDB recuperará automáticamente estos atributos de la tabla, lo que requiere que se lea todo el elemento de la tabla. Esto genera una latencia y unas operaciones de E/S adicionales que podrían evitarse.

Tenga en cuenta que, a menudo, las consultas "ocasionales" pueden convertirse en consultas "esenciales". Si hay atributos que no quiere proyectar porque prevé que solo los consultará ocasionalmente, piense en qué circunstancias esto podría cambiar y podría arrepentirse de no haberlos proyectado.

Para obtener más información sobre recuperaciones de tablas, consulte Consideraciones sobre el rendimiento aprovisionado para los índices secundarios locales.

Tener en cuenta los límites de tamaño de la colección de elementos al crear índices secundarios locales

Una colección de elementos contiene todos los elementos de una tabla y los índices secundarios globales que tienen la misma clave de partición. Ninguna colección de elementos puede superar los 10 GB, así que es posible que se agote el espacio para un determinado valor de clave de partición.

Cada vez que se agrega o actualiza un elemento de una tabla, DynamoDB actualiza todos los índices secundarios locales afectados. Si los atributos indexados se han definido en la tabla, los índices secundarios locales también aumentarán.

Cuando cree un índice secundario local, piense cuántos datos se van a escribir en él y cuántos de estos elementos de datos tendrán el mismo valor de clave de partición. Si prevé que la suma de los elementos de la tabla y del índice con un determinado valor de clave de partición podría ser superior a 10 GB, piense si debería evitar crear el índice.

Si no puede evitar crear el índice secundario local, tendrá que prever el límite de tamaño de la colección de elementos y adoptar medidas antes de superarlo. Se recomienda utilizar el parámetro `ReturnItemCollectionMetrics` al escribir los elementos para monitorear y alertar sobre los tamaños de las colecciones de elementos que se acerquen al límite de 10 GB. Si se supera el tamaño máximo de la colección de elementos, habrá intentos de escritura fallidos. Para mitigar los problemas de tamaño de las colecciones de elementos, monitoree los tamaños y emita alertas al respecto antes de que afecten a su aplicación.

> **Note**
>
> Una vez creado el índice secundario local ya no se puede eliminar.

Para obtener estrategias sobre cómo mantenerse dentro de los límites y adoptar medidas correctivas, consulte Límite del tamaño de una colección de elementos.

Sacar partido de los índices dispersos

En cualquier elemento de una tabla, DynamoDB escribe una entrada de índice solo si el valor de clave de ordenación del índice se encuentra presente en el elemento. Si la clave de clasificación no aparece en todos los elementos de la tabla, o si la clave de partición de índice no está presente en el elemento, se considera que el índice es disperso.

Los índices dispersos resultan útiles para las consultas que se realizan en una pequeña subsección de una tabla. Por ejemplo, supongamos que tiene una tabla en la que almacena todos los pedidos de los clientes con los siguientes atributos de clave:

- Clave de partición: : `CustomerId`
- Clave de clasificación: : `OrderId`

Para hacer un seguimiento de los pedidos abiertos, puede incluir un atributo llamado `isOpen` en los elementos de pedido que aún no se han enviado. Posteriormente, cuando estos pedidos se envíen, podrá borrar el atributo. Si crea un índice de `CustomerId` (clave de partición) e `isOpen` (clave de ordenación), en este índice solo aparecerán los pedidos en los que `isOpen` está definido. Cuando tiene miles de pedidos de los cuales solo hay abiertos un pequeño número, resulta más rápido y económico consultar los pedidos abiertos de ese índice que examinar toda la tabla.

En lugar de utilizar un tipo de atributo como `isOpen`, podría utilizar en el índice un atributo con un valor que diera como resultado un criterio de ordenación útil. Por ejemplo, podría utilizar un atributo `OrderOpenDate` establecido en la fecha en la que se realizó el pedido y borrarlo cuando el pedido se haya completado. De ese modo, cuando consulte el índice disperso, los elementos devueltos estarán ordenados en función de la fecha en que se realizó cada pedido.

Ejemplos de índices dispersos en DynamoDB

De forma predeterminada, los índices secundarios globales son dispersos. Cuando crea un índice secundario global, especifica una clave de partición y, de forma opcional, una clave de ordenación. Solo los elementos de la tabla principal que contienen esos atributos aparecen en el índice.

Si diseña un índice secundario global para que sea disperso, puede aprovisionarlo con un menor rendimiento de escritura que el de la tabla base y seguir disfrutando de un nivel de rendimiento excelente.

Por ejemplo, es posible que una aplicación de juego hiciera un seguimiento de todas las puntuaciones de cada usuario, pero lo normal es que solo deban consultarse unas cuentas puntuaciones. El diseño siguiente se adapta eficazmente a este escenario:

Table

Primary Key		Data Attributes...		
Partition Key	*Sort Key*			
Player_ID	Game_ID	Attribute 1	Attribute 2	Attribute 3
Rick	Game_1	Score: 36,750 *(game score)*	Date: 2017-11-14 *(date of game)*	
	Game_2	Score: 69,450 *(game score)*	Date: 2017-12-31 *(date of game)*	
	Game_3	Score: 135,900 *(game score)*	Date: 2018-01-19 *(date of game)*	Award: Champ *(type of award)*
Padma	Game_4	Score: 25,350 *(game score)*	Date: 2018-01-27 *(date of game)*	
	Game_5	Score: 69,450 *(game score)*	Date: 2028-01-19 *(date of game)*	
	Game_6	Score: 147,300 *(game score)*	Date: 2018-02-02 *(date of game)*	Award: Champ *(type of award)*
	Game_7	Score: 169,100 *(game score)*	Date: 2018-03-10 *(date of game)*	Award: Champ *(type of award)*

Aquí, Rick ha jugado tres partidas y ha conseguido el estado `Champ` en una de ellas. Padma ha jugado cuatro partidas y ha conseguido el estado `Champ` en dos de ellas. Observe que el atributo `Award` solamente está presente en los elementos en los que el usuario consiguió una recompensa. El índice secundario global asociado sería similar al siguiente:

GSI	Primary Key	Projected Attributes...			
	Partition Key				
	Award	Player_ID	Game_ID	Score	Date
	Champ	Rick	Game_3	135,900	2018-01-19
		Padma	Game_6	147,300	2018-02-02
		Padma	Game_7	169,100	2018-03-10

El índice secundario global únicamente contiene las puntuaciones máximas que se consultan con frecuencia, lo que conforma un pequeño subconjunto de los elementos de la tabla base.

Uso de índices secundarios globales para consultas de agregación materializadas en DynamoDB

Mantener métricas clave y agregaciones casi en tiempo real sobre datos que cambian rápidamente es algo que las compañías cada vez valoran más, ya que les permite tomar decisiones rápidas. Por ejemplo, una biblioteca de música quiere mostrar las canciones más descargadas prácticamente en tiempo real.

Pensemos en el diseño de la tabla de una biblioteca de música:

Music Library Table

Primary Key		Data-Item Attributes...			
Partition Key	Sort Key	Attribute 1	Attribute 2	Attribute 3	
Song-129 *(song ID)*	Details	Title: Wild Love *(song title)*	Artist: Argyboots *(artist or band name)*	Downloads: 15,314,822 *(lifetime total downloads)*	...etc.
	Month-2018-01	GSI Primary Key Month: 2018-01 *(download month)*	GSI Secondary Key MonthTotal: 1,746,992 *(month total downloads)*		
	Dld-9349823681	Time: 2018-01-01T00:00:07 *(download timestamp)*			
	Dld-9349823682	Time: 2018-01-01T00:00:07 *(download timestamp)*			
	Dld-9349823683	Time: 2018-01-01T00:00:07 *(download timestamp)*			

La tabla de este ejemplo almacena canciones con la clave de partición `songID`. Puede habilitar Amazon DynamoDB Streams en esta tabla y adjuntar una función Lambda a las transmisiones para que cada vez que se descargue una canción se agregue una entrada a la tabla con `Partition-`

`Key=SongID` y `Sort-Key=DownloadID`. A medida que se realizan, estas actualizaciones desencadenan una función Lambda en DynamoDB Streams. La función Lambda puede agregar y agrupar las descargas por `songID` y actualizar el elemento de nivel superior: `Partition-Key=songID` y `Sort-Key=Month`. Tenga en cuenta que, si una ejecución lambda falla justo después de escribir el nuevo valor agregado, esta ejecución podría volver a intentarse y el valor podría agregarse varias veces, lo que dejaría un valor aproximado.

Para leer las actualizaciones prácticamente en tiempo real con una latencia en milisegundos de un solo dígito, utilice el índice secundario global con las condiciones de consulta `Month=2018-01`, `ScanIndexForward=False`, `Limit=1`.

Otra importante optimización que se utiliza aquí es que el índice secundario global es un índice disperso y solo está disponible en los elementos que deben consultarse para recuperar los datos en tiempo real. El índice secundario global puede proporcionar flujos de trabajo adicionales que necesiten información sobre las 10 canciones más populares o cualquier canción descargada ese mes.

Sobrecarga de índices secundarios globales en DynamoDB

Aunque Amazon DynamoDB tiene una cuota predeterminada de 20 índices secundarios globales por tabla, en la práctica, se pueden indexar muchos más de 20 campos de datos. A diferencia de las tablas de los sistemas de administración de bases de datos relacionales (RDBMS, por sus siglas en inglés), donde el esquema es uniforme, en DynamoDB las tablas pueden contener diferentes tipo de elementos de datos a la vez. Además, un mismo atributo que se utilice en diferentes elementos puede contener tipos de información totalmente distinta.

Pensemos en el siguiente ejemplo, en el que se ha diseñado una tabla de DynamoDB donde se guarda una gran variedad de tipos de datos.

Primary Key		Data-Item Attributes...		
Partition Key	Sort Key	Attribute 1	Attribute 2	...
HR-974 *(employee ID)*	Employee_Name	Data: Murphy, John *(employee name)*	Start: 2008-11-08 *(start date)*	...etc.
	YYYY-Q1	Data: $5,477 *(order totals in USD)*	Name: Murphy, John *(employee name)*	
	HR_confidential	Data: 2008-11-08 *(hire date)*	Name: Murphy, John *(employee name)*	...etc.
	Warehouse_01	Data: Murphy, John *(employee name)*		
	v0_Job_title	Data: Operator-1 *(job title)*	Start: 2008-11-08 *(start date)*	...etc.
	v1_Job_title	Data: Operator-2 *(job title)*	Start: 2016-11-04 *(start date)*	...etc.
	v2_Job_title	Data: Supervisor-1 *(job title)*	Start: 2017-11-01 *(start date)*	...etc.

El atributo `Data`, que es común a todos los elementos, tiene un contenido que varía en función de su elemento principal. Si crea un índice secundario global para la tabla que utilice la clave de ordenación de la tabla como clave de partición y el atributo `Data` como clave de ordenación, puede realizar diferentes consultas utilizando el mismo índice secundario global. Estas consultas podrían ser las siguientes:

- Buscar un empleado por nombre en el índice secundario global utilizando `Employee_Name` como el valor de clave de partición y el nombre del empleado (por ejemplo `Murphy, John`) como valor de clave de ordenación.
- Utilizar el índice secundario global para buscar todos los empleados que trabajan en un determinado almacén a través del ID de almacén (por ejemplo, `Warehouse_01`).

- Obtener una lista de las contrataciones recientes consultando el índice secundario global con `HR_confidential` como valor de clave de partición y utilizando el rango de fechas como valor de clave de ordenación.

Uso de la partición de escritura del índice secundario global para las consultas de tabla selectivas en DynamoDB

Cuando necesita consultar datos recientes en un intervalo de tiempo específico, el requisito de DynamoDB de proporcionar una clave de partición para la mayoría de las operaciones de lectura puede suponer un reto. Para abordar este escenario, puede implementar un patrón de consulta eficaz mediante una combinación de partición de escritura y un índice secundario global (GSI).

Este enfoque le permite recuperar y analizar de manera eficiente datos confidenciales sin realizar análisis completos de tablas, que pueden requerir muchos recursos y ser costosos. Mediante el diseño estratégico de la estructura e indexación de la tabla, puede crear una solución flexible que admita la recuperación de datos basada en el tiempo, al mismo tiempo que mantiene un rendimiento óptimo.

Temas

- Diseño de patrón
- Estrategia de particiones
- Consulta del GSI particionado
- Consideraciones sobre la ejecución de consultas paralelas
- Ejemplo de código

Diseño de patrón

Cuando trabaje con DynamoDB, puede superar los desafíos de recuperación de datos basados en el tiempo mediante la implementación de un patrón sofisticado que combina particiones de escritura e índices secundarios globales para habilitar consultas flexibles y eficientes en ventanas de datos recientes.

Estructura de la tabla

- Clave de partición (PK): “Username”

Estructura del GSI

- Clave de partición de GSI (PK_GSI): “ShardNumber#”
- Clave de clasificación de GSI (SK_GSI): marca temporal ISO 8601 (por ejemplo, “2030-04-01T12:00:00Z”)

[TABLE] - TimeBounded Metadata Actions

PK	SK_GSI	Address	PK_GSI
Trenton69	2023-12-05T06:40:31.312Z	117 Hudson Divide	ShardNumber#0
Scot_Langworth-Prosacco	2024-10-09T14:44:45.819Z	84021 Herzog Skyway	ShardNumber#6
Juliet.Morissette	2025-03-28T07:20:56.538Z	4871 N Broadway	ShardNumber#4
Kay.Von71	2024-11-23T16:19:26.763Z	14554 Odell Throughway	ShardNumber#2
Kiarra43	2023-11-15T16:21:57.078Z	8471 London Road	ShardNumber#1
Marcella91	2024-12-17T09:02:31.623Z	10271 Nick Crescent	ShardNumber#6
Bernard.Lemke36	2025-09-09T00:18:34.482Z	5438 Douglas Groves	ShardNumber#2
Daisha83	2024-01-24T06:38:08.482Z	8786 Armstrong Radial	ShardNumber#3
Marcos_Schmitt58	2024-04-06T17:38:58.551Z	510 S Center Street	ShardNumber#3
Jeremie_VonRueden-Mills	2025-04-06T04:24:23.701Z	157 Koch Drives	ShardNumber#1
Verna28	2025-10-10T18:42:21.059Z	70040 Baylee Trafficway	ShardNumber#1
Trycia.Doyle	2024-01-10T17:00:08.360Z	9511 Tillman Extensions	ShardNumber#2

Estrategia de particiones

Suponiendo que decida utilizar diez particiones, los números de partición podrían estar en un intervalo del cero al nueve. Al registrar una actividad, debe calcular el número de partición (por ejemplo, mediante una función de hash en el ID de usuario y, a continuación, tomando el módulo del número de particiones) y anteponerlo a la clave de partición de GSI. Este método distribuye las entradas entre diferentes particiones, lo que mitiga el riesgo de particiones activas.

Consulta del GSI particionado

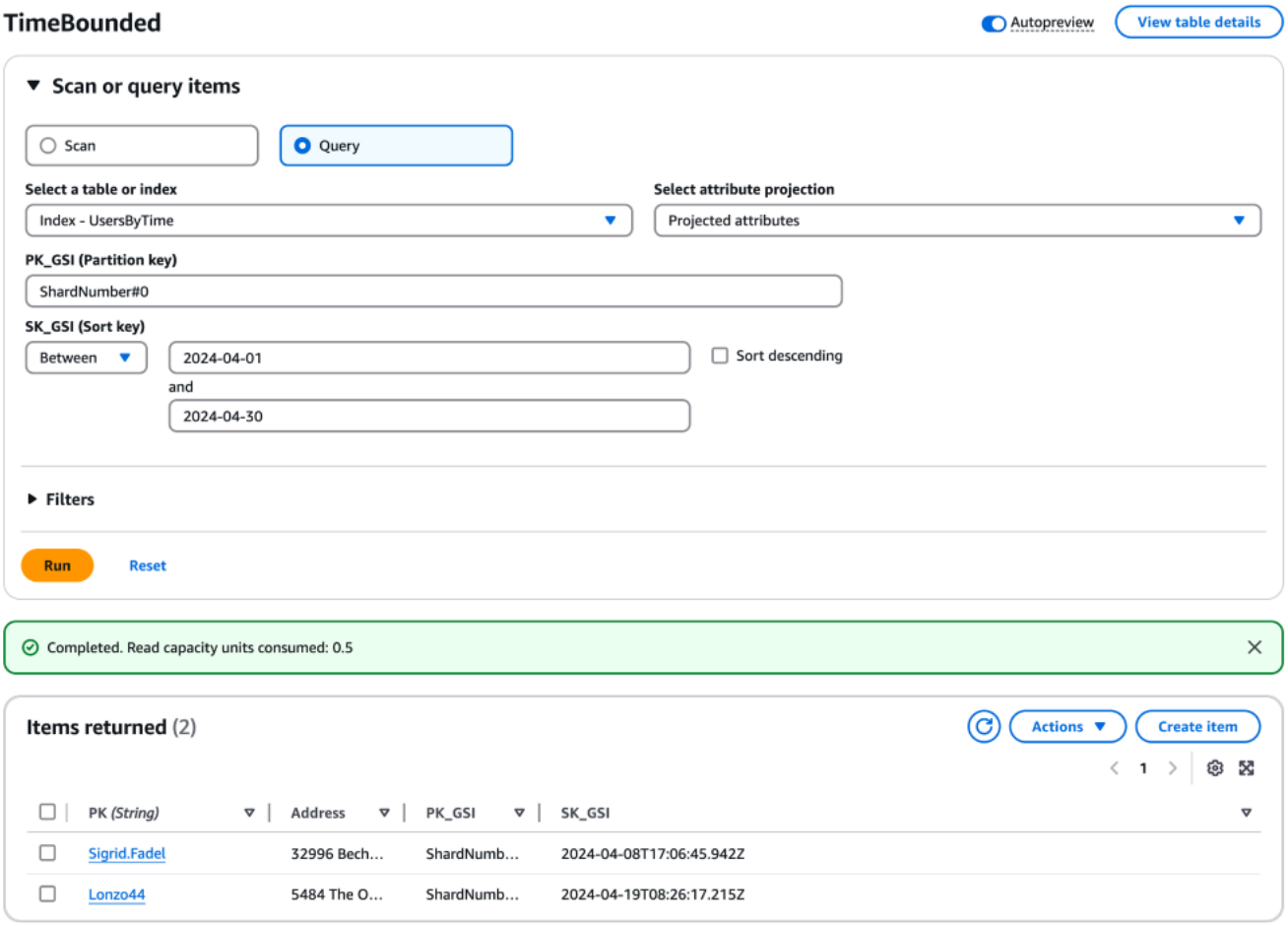
La consulta en todos los fragmentos de elementos en un intervalo de tiempo concreto en una tabla de DynamoDB, donde los datos se fragmentan en varias claves de partición, requiere un enfoque diferente al de la consulta en una sola partición. Como las consultas de DynamoDB están limitadas a una sola clave de partición a la vez, no puede consultar directamente varias particiones con una sola

operación de consulta. No obstante, puede lograr el resultado deseado mediante la lógica en el nivel de aplicación; para ello, realice varias consultas, cada una de ellas dirigida a una partición específica y, a continuación, agregue los resultados. En el siguiente procedimiento se explica cómo hacerlo.

Consulta y agregación de particiones

1. Identifique el intervalo de números de particiones utilizado en la estrategia de partición. Por ejemplo, si tiene diez particiones, los números de partición estarán en el intervalo del cero al nueve.
2. Para cada partición, construya y ejecute una consulta para recuperar elementos en el intervalo de tiempo deseado. Estas consultas pueden ejecutarse en paralelo para mejorar la eficiencia. Utilice la clave de partición con el número de partición y la clave de clasificación con el intervalo de tiempo para estas consultas. Aquí tiene un ejemplo de consulta para una sola partición:

```
aws dynamodb query \
    --table-name "YourTableName" \
    --index-name "YourIndexName" \
    --key-condition-expression "PK_GSI = :pk_val AND SK_GSI BETWEEN :start_date
 AND :end_date" \
    --expression-attribute-values '{
        ":pk_val": {"S": "ShardNumber#0"},
        ":start_date": {"S": "2024-04-01"},
        ":end_date": {"S": "2024-04-30"}
    }'
```



Debería replicar esta consulta para cada partición, ajustando la clave de partición en consecuencia (por ejemplo, “ShardNumber#1”, “ShardNumber#2”, ..., “ShardNumber#9”).

3. Agregue los resultados de cada consulta una vez que todas las consultas estén completas. Realice esta agregación en el código de la aplicación; para ello, combine los resultados en un único conjunto de datos que represente los elementos de todas las particiones en el intervalo de tiempo especificado.

Consideraciones sobre la ejecución de consultas paralelas

Cada consulta consume capacidad de lectura de la tabla o el índice. Si utiliza rendimiento aprovisionado, asegúrese de que la tabla esté aprovisionada con capacidad suficiente para manejar la ampliación de consultas paralelas. Si utiliza la capacidad bajo demanda, tenga en cuenta las posibles implicaciones de costos.

Ejemplo de código

Para ejecutar consultas paralelas en particiones de DynamoDB con Python, puede utilizar la biblioteca boto3, que es el SDK de Amazon Web Services para Python. En este ejemplo se supone que tiene boto3 instalado y configurado con las credenciales de AWS adecuadas.

En el siguiente código Python se muestra cómo realizar consultas paralelas en varias particiones para un intervalo de tiempo determinado. Utiliza futuros concurrentes para ejecutar consultas en paralelo, lo que reduce el tiempo total de ejecución en comparación con la ejecución secuencial.

```
import boto3
from concurrent.futures import ThreadPoolExecutor, as_completed

# Initialize a DynamoDB client
dynamodb = boto3.client('dynamodb')

# Define your table name and the total number of shards
table_name = 'YourTableName'
total_shards = 10  # Example: 10 shards numbered 0 to 9
time_start = "2030-03-15T09:00:00Z"
time_end = "2030-03-15T10:00:00Z"

def query_shard(shard_number):
    """
    Query items in a specific shard for the given time range.
    """
    response = dynamodb.query(
        TableName=table_name,
        IndexName='YourGSIName',  # Replace with your GSI name
        KeyConditionExpression="PK_GSI = :pk_val AND SK_GSI BETWEEN :date_start
 AND :date_end",
        ExpressionAttributeValues={
            ":pk_val": {"S": f"ShardNumber#{shard_number}"},
            ":date_start": {"S": time_start},
            ":date_end": {"S": time_end},
        }
    )
    return response['Items']

# Use ThreadPoolExecutor to query across shards in parallel
with ThreadPoolExecutor(max_workers=total_shards) as executor:
    # Submit a future for each shard query
```

```
    futures = {executor.submit(query_shard, shard_number): shard_number for
 shard_number in range(total_shards)}

    # Collect and aggregate results from all shards
    all_items = []
    for future in as_completed(futures):
        shard_number = futures[future]
        try:
            shard_items = future.result()
            all_items.extend(shard_items)
            print(f"Shard {shard_number} returned {len(shard_items)} items")
        except Exception as exc:
            print(f"Shard {shard_number} generated an exception: {exc}")

# Process the aggregated results (e.g., sorting, filtering) as needed
# For example, simply printing the count of all retrieved items
print(f"Total items retrieved from all shards: {len(all_items)}")
```

Antes de ejecutar este código, asegúrese de reemplazar `YourTableName` y `YourGSIName` por los nombres reales de la tabla y GSI de la configuración de DynamoDB. Además, ajuste las variables `total_shards`, `time_start` y `time_end` de acuerdo con los requisitos específicos de su empresa.

Este script consulta cada partición en busca de elementos en el intervalo de tiempo especificado y agrega los resultados.

Uso de índices secundarios globales para crear una réplica eventualmente consistente en DynamoDB

Puede utilizar un índice secundario global para crear una réplica eventualmente consistente La creación de una réplica permite realizar las siguientes tareas:

- Configure diferentes capacidades de lectura aprovisionada para diferentes lectores. Por ejemplo, supongamos que tenemos dos aplicaciones: una aplicación controla las consultas de alta prioridad y necesita el máximo nivel de rendimiento de lectura, mientras que la otra controla las consultas de baja prioridad que pueden tolerar la limitación controlada de la actividad de lectura.

  Si ambas aplicaciones leen desde la misma tabla, una carga de lectura pesada de la aplicación de baja prioridad podría consumir toda la capacidad de lectura disponible para la tabla. Esto limitaría la actividad de lectura de la aplicación de alta prioridad.

En su lugar, puede crear una réplica a través de un índice secundario global cuya capacidad de lectura la puede establecer separada de la de la tabla en sí. A continuación, puede hacer que la aplicación de prioridad baja consulte la réplica en lugar de la tabla.

- Elimine las lecturas de una tabla por completo. Por ejemplo, puede tener una aplicación que capture un alto volumen de actividad de clics de un sitio web y no quiere correr el riesgo de que las lecturas interfieran con ello. Puede aislar esta tabla e impedir las lecturas de otras aplicaciones (consulte Uso de condiciones de las políticas de IAM para control de acceso preciso), mientras que permite que otras aplicaciones lean una réplica creada usando un índice secundario global.

Para crear una réplica, configure un índice secundario global que tenga el mismo esquema de claves que la tabla principal y proyectar en él algunos (o todos) los atributos sin clave. En aplicaciones, puede dirigir parte (o la totalidad) de la actividad de lectura a este índice secundario global, en lugar de a la tabla principal. A continuación, puede ajustar la capacidad de lectura aprovisionada del índice secundario global para manejar esas lecturas sin cambiar la capacidad de lectura aprovisionada de la tabla principal.

Siempre se produce un pequeño retraso de propagación entre el momento en que se escribe en la tabla principal y el momento en que los datos escritos aparecen en el índice. En otras palabras, sus aplicaciones deben tener en cuenta que la réplica del índice secundario global solo es eventualmente consistente con la tabla principal.

Puede crear varias réplicas de índice secundario global para admitir diferentes patrones de lectura. Cuando cree las réplicas, proyecte sólo los atributos que cada patrón de lectura requiere realmente. Como resultado, una aplicación puede consumir menos capacidad de lectura aprovisionada para obtener solo los datos que necesita en lugar de tener que leer el elemento de la tabla principal. Esta optimización puede suponer un importante ahorro en coste a largo plazo.

Prácticas recomendadas para almacenar elementos y atributos grandes en DynamoDB

Amazon DynamoDB limita el tamaño de cada uno de los elementos que se almacenan en una tabla a 400 KB (consulte Cuotas en Amazon DynamoDB). Si la aplicación necesita almacenar más datos en un elemento de lo que permite el límite de tamaño de DynamoDB, puede intentar comprimir uno o varios atributos grandes o dividir el elemento en varios elementos (indizados eficazmente mediante claves de ordenación). También puede almacenar el elemento como un objeto de Amazon Simple

Storage Service (Amazon S3) y almacenar el identificador de objeto de Amazon S3 en el elemento de DynamoDB.

Se recomienda utilizar el parámetro `ReturnConsumedCapacity` al escribir los elementos para monitorear y alertar sobre los tamaños de los elementos que se acerquen al límite máximo de 400 KB. Si se supera el tamaño máximo del elemento, habrá intentos de escritura fallidos. DynamoDB devolverá un error ValidationException. Para mitigar los problemas de tamaño de los elementos, monitoree los tamaños y emita alertas al respecto antes de que afecten a su aplicación.

Compresión de valores de atributos grandes

Si los valores de atributo grandes se comprimen, es posible que se ajusten a los límites de los elementos de DynamoDB y se reduzcan los costes de almacenamiento. Los algoritmos de compresión, como GZIP o LZO, generan una salida binaria que se puede almacenar en un tipo de atributo `Binary` dentro del elemento.

Por ejemplo, piense en una tabla donde se almacenan los mensajes escritos por los usuarios de un foro. Estos mensajes suelen contener largas cadenas de texto que pueden comprimirse. Si bien la compresión puede reducir el tamaño de los elementos, la desventaja es que no se pueden filtrar los valores comprimidos de los atributos.

Para ver un código de muestra en el que se ilustra cómo comprimir este tipo de mensajes en DynamoDB, consulte lo siguiente:

- Ejemplo: control de atributos de tipo binario mediante la API de documentos de AWS SDK para Java
- Ejemplo: control de atributos de tipo binario mediante la API de bajo nivel de AWS SDK para .NET

Particionamiento vertical

Una solución alternativa para gestionar elementos grandes es partirlos en fragmentos de datos más pequeños y asociar todos los elementos relevantes por el valor de la clave de partición. A continuación, se puede utilizar una cadena de claves de clasificación para identificar la información asociada que se almacena en ella. Al hacer esto y agrupar varios elementos por el mismo valor de clave de partición, se crea una colección de elementos.

Para obtener más información acerca de este enfoque, consulte las publicaciones del blog:

- Use vertical partitioning to scale data efficiently in Amazon DynamoDB

- Implement vertical partitioning in Amazon DynamoDB using AWS Glue

Almacenamiento de valores de atributos grandes en Amazon S3

Como se indicó anteriormente, también puede usar Amazon S3 para almacenar valores de atributos grandes que no caben en un elemento de DynamoDB. Puede almacenar estos atributos como un objeto de Amazon S3 y almacenar después el identificador del objeto en el elemento de DynamoDB.

También puede aprovechar la compatibilidad con los metadatos de objetos de Amazon S3 para proporcionar un enlace de regreso al elemento principal de DynamoDB. Almacene el valor de clave principal del elemento como metadatos de Amazon S3 del objeto en Amazon S3. Normalmente, esto contribuye positivamente al mantenimiento de los objetos de Amazon S3.

Por ejemplo, veamos la tabla `ProductCatalog`. En los elementos de esta tabla, se almacena información sobre el precio del elemento, su descripción, los autores (en el caso de los libros) y las dimensiones (en el caso de otros productos). Si quisiera almacenar una imagen de cada producto que fuera demasiado grande como para que cupiera en un elemento, podría almacenar las imágenes en Amazon S3 en lugar de en DynamoDB.

Si utiliza esta estrategia, tenga en cuenta lo siguiente:

- DynamoDB no admite transacciones que atraviesen Amazon S3 y DynamoDB. Por lo tanto, la aplicación debe solventar cualquier error, que podría estar relacionado con la limpieza de objetos de Amazon S3 huérfanos.
- Amazon S3 impone un límite sobre la longitud de los identificadores de objetos. Por tanto, debe organizar los datos de forma que no generen identificadores de objetos excesivamente largos ni infrinjan otras restricciones de Amazon S3.

Para obtener más información acerca de cómo utilizar Amazon S3, consulte la Guía del usuario de Amazon Simple Storage Service.

Prácticas recomendadas para administrar los datos de serie temporal en DynamoDB

Los principios generales de diseño de Amazon DynamoDB recomiendan utilizar la menor cantidad de tablas posible. En la mayoría de las aplicaciones, solo se necesita una tabla. Sin embargo, para los

datos de series temporales, a menudo lo mejor para administrarlos es usar una tabla por aplicación y periodo.

Patrón de diseño de los datos de serie temporal

Imagine un caso típico de una serie temporal en el que quiere hacer un seguimiento de una gran cantidad de eventos. Tiene un patrón de acceso de escritura que establece que se registren todos los eventos con la fecha de hoy. El patrón de acceso de lectura podría establecer que los eventos de hoy se lean con más frecuencia, que los eventos de ayer se lean con mucha menos frecuencia y que los eventos más antiguos apenas se lean. Una manera de administrarlo consiste en incorporar la fecha y hora actuales en la clave principal.

Normalmente, el siguiente patrón de diseño sirve para administrar este tipo de escenarios eficazmente:

- Cree una tabla por periodo, aprovisionada con la capacidad requerida de lectura y escritura y con los índices que se necesitan.
- Antes de que termine cada periodo, precompile la tabla para el siguiente periodo. Justo cuando termine el período actual, dirija el tráfico de los eventos a la nueva tabla. Puede asignar nombres a estas tablas que indiquen los períodos que contienen.
- Tan pronto como la tabla deje de estar disponible para escribir en ella, reduzca su capacidad de escritura aprovisionada a un valor menor (por ejemplo, 1 WCU) y aprovisione la capacidad de lectura apropiada, según proceda. Reduzca la capacidad de lectura aprovisionada de las tablas anteriores a medida que vayan venciendo. Puede optar por archivar o eliminar las tablas cuyo contenido va a necesitar en pocas ocasiones o no va a necesitar nunca.

Se trata de asignar, para el periodo actual, los recursos requeridos que vayan a experimentar el máximo volumen de tráfico y de reducir el aprovisionamiento de las tablas más antiguas que no se utilizan activamente, con lo que se ahorra en costos. En función de sus necesidades de negocio, podría plantearse fragmentar la escritura con el fin de distribuir el tráfico de manera uniforme en la clave de partición lógica. Para obtener más información, consulte Uso de la partición de escritura para distribuir las cargas de trabajo uniformemente en la tabla de DynamoDB.

Ejemplos de tablas de serie temporal

A continuación se muestra un ejemplo de datos de serie temporal. En él, la tabla actual está aprovisionada con una capacidad de lectura/escritura mayor y las tablas anteriores se han reducido, porque el acceso a ellas es infrecuente:

Current table — Provisioned at: WCU=750 and RCU=300

Primary Key		Attributes		
Partition Key	Sort Key	Radiant Intensity	Wavelength	...
2018-03-15	00:00:00.002	17.372 W/Sr	713 nm	...
2018-03-15	00:00:00.004	17.385 W/Sr	712 nm	...
2018-03-15	00:00:00.005	17.478 W/Sr	708 nm	...
2018-03-15	00:00:00.007	19.172 W/Sr	674 nm	...
• • •	• • •	• • •	• • •	...

Previous table — Provisioned at: WCU=1 and RCU=100

Primary Key		Attributes		
Partition Key	Sort Key	Radiant Intensity	Wavelength	...
2018-03-14	00:00:00.001	16.473 W/Sr	512	...
2018-03-14	00:00:00.003	16.489 W/Sr	519	...
2018-03-14	00:00:00.004	16.814 W/Sr	522	...
2018-03-14	00:00:00.006	16.719 W/Sr	506	...
• • •	• • •	• • •	• • •	...

Older table — Provisioned at: WCU=1 and RCU=1

Primary Key		Attributes		
Partition Key	Sort Key	Radiant Intensity	Wavelength	...
2018-03-10	00:00:00.001	13.669 W/Sr	456	...
2018-03-10	00:00:00.002	13.522 W/Sr	459	...
2018-03-10	00:00:00.004	13.596 W/Sr	457	...
2018-03-10	00:00:00.005	15.721 W/Sr	425	...
• • •	• • •	• • •	• • •	...

Prácticas recomendadas para administrar las relaciones de varios a varios en las tablas de DynamoDB

Las listas de adyacencia conforman un patrón de diseño que resulta muy útil para modelar relaciones de varios a varios en Amazon DynamoDB. En otras palabras, constituyen un mecanismo para representar datos gráficos (nodos y bordes) en DynamoDB.

Patrón de diseño de listas de adyacencia

Cuando varias entidades de una aplicación tienen una relación de varios a varios entre ellas, la relación puede modelarse como una lista de adyacencia. En este patrón, todas las entidades del nivel superior (que se corresponderían con los nodos en el modelo gráfico) se representan utilizando la clave de partición. Cualquier relación con otras entidades (los límites en el modelo gráfico) se representa como un elemento dentro de la partición, donde el valor de la clave de ordenación se establece en el ID de la entidad de destino (nodo de destino).

Algunas de las ventajas de este patrón son la escasa duplicación de los datos y la simplificación de los patrones de consulta para buscar todas las entidades (nodos) relacionadas con una entidad de destino (que tiene un límite con un nodo de destino).

Un ejemplo real en el que este patrón resulta útil son los sistemas de facturación en los que las facturas (invoices) contienen varias cuentas (bills). Una cuenta puede ir en varias facturas. La clave de partición en este ejemplo es `InvoiceID` o bien `BillID`. Las particiones `BillID` tienen todos los atributos específicos de las facturas. Las particiones `InvoiceID` tienen un elemento que almacena atributos específicos de las facturas y un elemento para cada `BillID` que aparece en la factura.

El esquema sería similar al siguiente:

Table	Primary Key		Data Attributes...	
	Partition Key	*Sort Key (and GSI PK)*		
	`Invoice-92551`	**Inv_ID:** Invoice-92551 *(invoice ID)*	**Dated:** 2018-02-07 *(date created)*	More attributes of this invoice...
		Bill_ID: Bill-4224663 *(bill ID)*	**Dated:** 2017-12-03 *(date created)*	Attributes of this bill *in this invoice*..
		Bill_ID: Bill-4224687 *(bill ID)*	**Dated:** 2018-01-09 *(date created)*	Attributes of this bill *in this invoice*..
	`Invoice-92552`	**Inv_ID:** Invoice-92552 *(invoice ID)*	**Dated:** 2018-03-04 *(date created)*	More attributes of this invoice...
		Bill_ID: Bill-4224687 *(bill ID)*	**Dated:** 2018-01-09 *(date created)*	Attributes of this bill *in this invoice*..
	`Bill-4224663`	**Bill_ID:** Bill-4224663 *(bill ID)*	**Dated:** 2017-12-03 *(date created)*	More attributes of this bill...
	`Bill-4224687`	**Bill_ID:** Bill-4224687 *(bill ID)*	**Dated:** 2018-01-09 *(date created)*	More attributes of this bill...

En el esquema anterior, puede ver que las cuentas de una factura pueden consultarse utilizando la clave principal de la tabla. Para buscar todas las facturas que contienen parte de una cuenta, cree un índice secundario global de la clave de ordenación de la tabla.

Las proyecciones del índice secundario global serían similares a las siguientes.

GSI	Primary Key *Partition Key*	Projected Attributes...	
	Bill-4224663	**Bill_ID:** Bill-4224663 *(table primary key)*	Attributes of this bill...
		Inv_ID: Invoice-92551 *(table primary key)*	Attributes of this bill *in this invoice*..
	Bill-4224687	**Bill_ID:** Bill-4224687 *(table primary key)*	Attributes of this bill...
		Inv_ID: Invoice-92551 *(table primary key)*	Attributes of this bill *in this invoice*..
		Inv_ID: Invoice-92552 *(table primary key)*	Attributes of this bill *in this invoice*..
	Invoice-92551	**Inv_ID:** Invoice-92551 *(table primary key)*	Attributes of this invoice...
	Invoice-92552	**Inv_ID:** Invoice-92552 *(table primary key)*	Attributes of this invoice...

Patrón de gráficos materializados

Muchas aplicaciones se crean en torno a la información que se tiene sobre las clasificaciones entre homólogos, las relaciones comunes entre entidades, el estado de las entidades adyacentes y otros tipos de flujos de trabajo gráficos. En estos tipos de aplicaciones, considere la posibilidad de utilizar el siguiente patrón de diseño de esquemas.

TABLE	Primary Key		Attributes		
	PK (NodeId)	SK (TypeTarget, GSI 2 SK)	Data	GSI PK	Graph Projections
	1	DATE|2|BIRTH	1980-12-19	Hash(Person.Data)	
		PERSON|1	Data (GSI1 SK)	GSI PK	
			John Doe	Hash(Person.Data)	
		PERSON|5|FRIEND	Data	GSI PK	
			Jane Smith	Hash(Person.Data)	
		PLACE|4|BIRTH	Data	GSI PK	
			USA|Texas|Austin	Hash(Person.Data)	
		SKILL|6	Data	GSI PK	
			Java Developer|Senior	Hash(Person.Data)	
	2	DATE|2	Data	GSI PK	
			1980-12-19	0	
	3	PLACE|3	Data	GSI PK	
			UK|England|London	0	
	4	PLACE|4	Data	GSI PK	
			USA|Texas|Austin	0	...
	5	DATE|2|BIRTH	Data	GSI PK	
			1980-12-19	Hash(Person.Data)	
		PERSON|5	Data	GSI PK	
			Jane Smith	Hash(Person.Data)	
		PERSON|1|FRIEND	Data	GSI PK	
			John Doe	Hash(Person.Data)	
		PLACE|3|BIRTH	Data	GSI PK	
			UK|England|London	Hash(Person.Data)	
		SKILL|7	Data	GSI PK	
			Guitar|Advanced	Hash(Person.Data)	
	6	SKILL|6	Data	GSI PK	
			Java Developer	0	
	7	SKILL|7	Data	GSI PK	
			Guitar	0	

	Primary Key		Attributes		
	GSI PK	GSI 1 SK (Data)			
GSI 1	O-N	1980-12-19	NodeID	TypeTarget	Graph Projections
			2	DATE|2	...
			NodeID	TypeTarget	
			1	DATE|2|BIRTH	
			NodeID		
			5		
		Guitar	NodeID	TypeTarget	
			7	SKILL|7	
		Guitar|Advanced	NodeID		
			5		
		Jane Smith	NodeID	TypeTarget	
			5	Person|5	
			NodeID	TypeTarget	
			1	Person|5|FRIEND	
		Java Developer	NodeID	TypeTarget	
			6	SKILL|6	
		Java Developer|Senior	NodeID		
			1		
		John Doe	NodeID	TypeTarget	
			1	Person|1	
			NodeID	TypeTarget	
			5	Person|1|FRIEND	
		UK|England|London	NodeID	TypeTarget	
			3	PLACE|3	
			NodeID	TypeTarget	
			1	PLACE|3|BIRTH	
		USA|Texas|Austin	NodeID	TypeTarget	
			4	PLACE|4	
			NodeID	TypeTarget	
			5	PLACE|4|BIRTH	

	Primary Key		Attributes		
	GSI PK	GSI 2 SK (TypeTarget)			
GSI 2	O-N	DATE|2	NodeID	Data	Graph Projections
			2	1980-12-19	...
		DATE|2|BIRTH	NodeID		
			1		
			NodeID		
			5		
		PERSON|1	NodeID	Data	
			1	John Doe	
		PERSON|1|FRIEND	NodeID		
			5		
		PERSON|5	NodeID	Data	
			5	Jane Smith	
		PERSON|5|FRIEND	NodeID		
			1		
		PLACE|3	NodeID	Data	
			3	UK|England|London	
		PLACE|3|BIRTH	NodeID		
			5		
		PLACE|4	NodeID	Data	
			4	USA|texas|Austin	
		PLACE|4|BIRTH	NodeID		
			1		
		SKILL|6	NodeID	Data	
			6	Java Developer	
			NodeID	Data	
			1	Java Developer|Senior	
		SKILL|7	NodeID	Data	
			7	Guitar	
			NodeID	Data	
			5	Guitar|Advanced	

En el esquema anterior, se muestra una estructura de datos gráficos definida por un conjunto de particiones de datos que contienen los elementos que definen los límites y los nodos del gráfico. Los elementos de límite contienen un atributo `Target` y `Type`. Estos atributos se utilizan como parte de una clave compuesta llamada "TypeTarget" para identificar el elemento de una partición de la tabla principal o de un índice secundario global.

El primer índice secundario global se basa en el atributo `Data`. Este atributo utiliza la sobrecarga de índices secundarios globales que se describió anteriormente para indexar varios tipos de atributos diferentes; en este caso, `Dates`, `Names`, `Places` y `Skills`. Aquí, un índice secundario global es capaz de indexar eficazmente cuatro atributos diferentes.

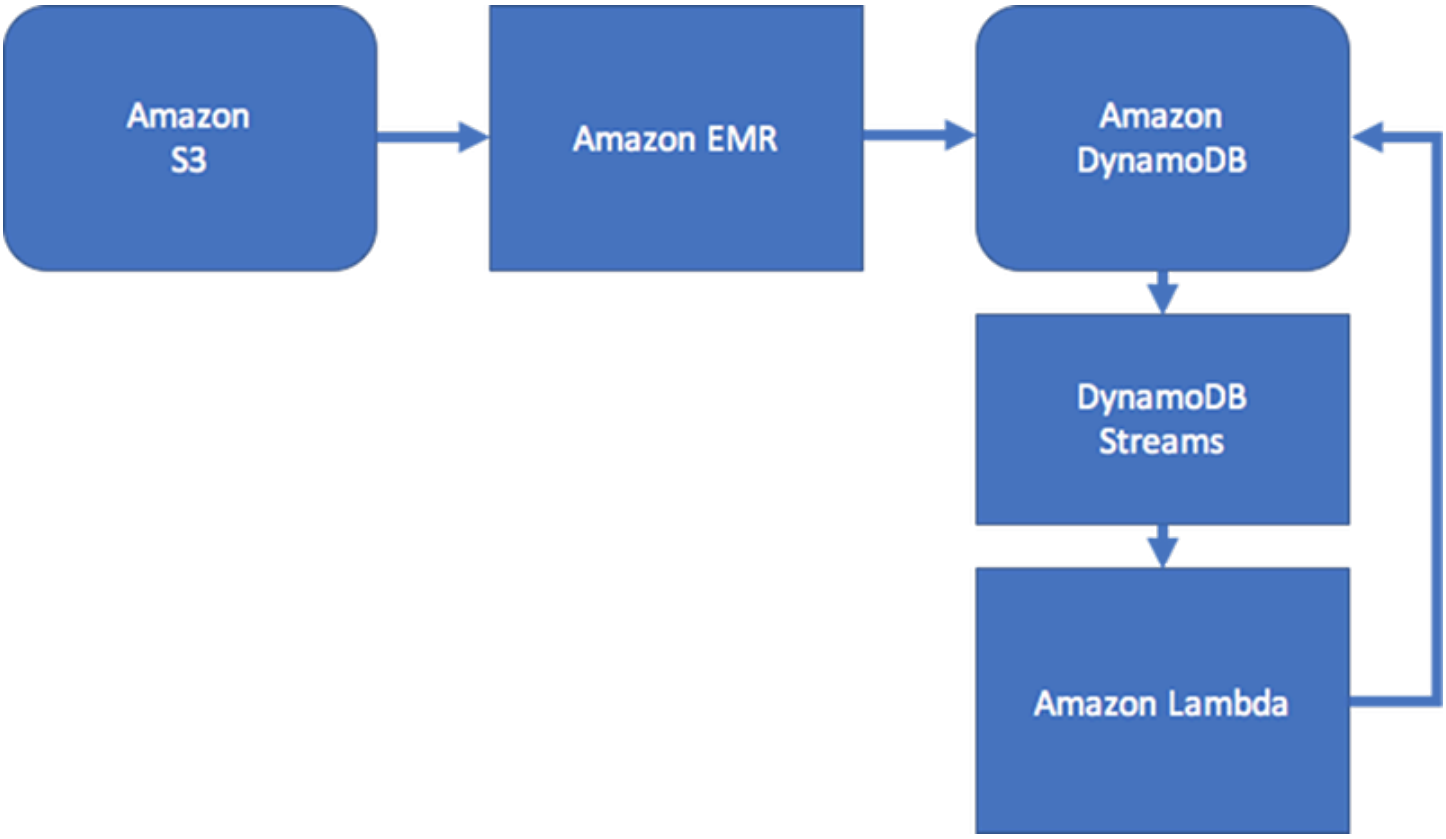
A medida que inserta elementos en la tabla, puede utilizar una estrategia de fragmentación inteligente para distribuir conjuntos de elementos con grandes agregaciones (Birthdate, Skill) entre tantas particiones lógicas de los índices secundarios globales como sean necesarias y evitar así el problema de que se creen puntos de lectura o escritura "calientes".

El resultado de esta combinación de patrones de diseño es un almacén de datos sólido para flujos de trabajo gráficos en tiempo real de gran eficacia. Estos flujos de trabajo pueden proporcionar

consultas de agregación de estado y límite de entidades adyacentes de gran rendimiento para motores de recomendaciones, aplicaciones de redes sociales, clasificaciones de nodos, agregaciones de subárboles y otros casos de uso gráficos habituales.

Si en el caso de uso no es importante la coherencia de datos en tiempo real, puede utilizar un proceso de Amazon EMR programado para rellenar los bordes con agregaciones de resumen de gráficos que sean relevantes para los flujos de trabajo. Si la aplicación no necesita saber inmediatamente cuándo se agrega un límite al gráfico, pude utilizar un proceso programado para agregar resultados.

Para mantener cierto nivel de consistencia, el diseño podría incluir Amazon DynamoDB Streams y AWS Lambda a la hora de procesar actualizaciones de borde. También podría usar un trabajo de Amazon EMR para validar los resultados a intervalos periódicos. Ente enfoque se ilustra en el siguiente diagrama. Normalmente, se emplea en aplicaciones de redes sociales, donde el costo de una consulta en tiempo real es elevado y la necesidad de conocer inmediatamente las actualizaciones de cada usuario es baja.



Por lo general, las aplicaciones de seguridad y administración de servicios de TI (ITSM) tienen que responder en tiempo real a los cambios de estado de las entidades que se componen de agregaciones de límites complejas. Este tipo de aplicaciones necesitan un sistema que pueda admitir

varias agregaciones en tiempo real de relaciones de segundo y tercer nivel en nodos o recorridos de límites complejos. Si el caso de uso requiere estos tipos de flujos de trabajo de consultas de gráficos en tiempo real, le recomendamos que considere la posibilidad de utilizar Amazon Neptune para administrarlos.

> **Note**
>
> Si necesita consultar conjuntos de datos muy conectados o ejecutar consultas que deban recorrer varios nodos (también conocidas como consultas multisalto) con una latencia de milisegundos, debería considerar la posibilidad de utilizar Amazon Neptune. Amazon Neptune es un motor de base de datos de gráficos de alto rendimiento, optimizado para almacenar miles de millones de relaciones y consultar el gráfico con una latencia de milisegundos.

Prácticas recomendadas para consultar y analizar datos en DynamoDB

En esta sección se explican algunas prácticas recomendadas relativas a las operaciones `Query` y `Scan` en Amazon DynamoDB.

Consideraciones sobre el rendimiento de los análisis

En general, las operaciones `Scan` son menos eficientes que otras operaciones de DynamoDB. Una operación `Scan` siempre analiza toda la tabla o el índice secundario. Luego filtra los valores para obtener el resultado deseado, para lo cual se agrega el paso adicional de eliminar los datos del conjunto de resultados.

Si es posible, evite usar la operación `Scan` en una tabla o un índice de gran tamaño con un filtro que elimine muchos resultados. Además, a medida que la tabla o el índice aumentan de tamaño, la operación `Scan` resulta más lenta. La operación `Scan` examina cada elemento para comprobar si presenta los valores solicitados y permite utilizar el rendimiento aprovisionado para una tabla o un índice grandes en una sola operación. Para lograr tiempos de respuesta más breves, diseñe las tablas o los índices de tal forma que las aplicaciones puedan utilizar `Query` en lugar de `Scan`. (Para las tablas, considere también la posibilidad de usar las API `GetItem` y `BatchGetItem`).

De forma alternativa, puede diseñar la aplicación para usar operaciones `Scan` de tal forma que minimicen el impacto en la tasa de solicitudes. Esto puede incluir el modelado cuando sería más

eficiente utilizar un índice secundario global en lugar de una operación `Scan`. Encontrará más información sobre este proceso en el siguiente vídeo.

Modelado de patrones de acceso de baja velocidad

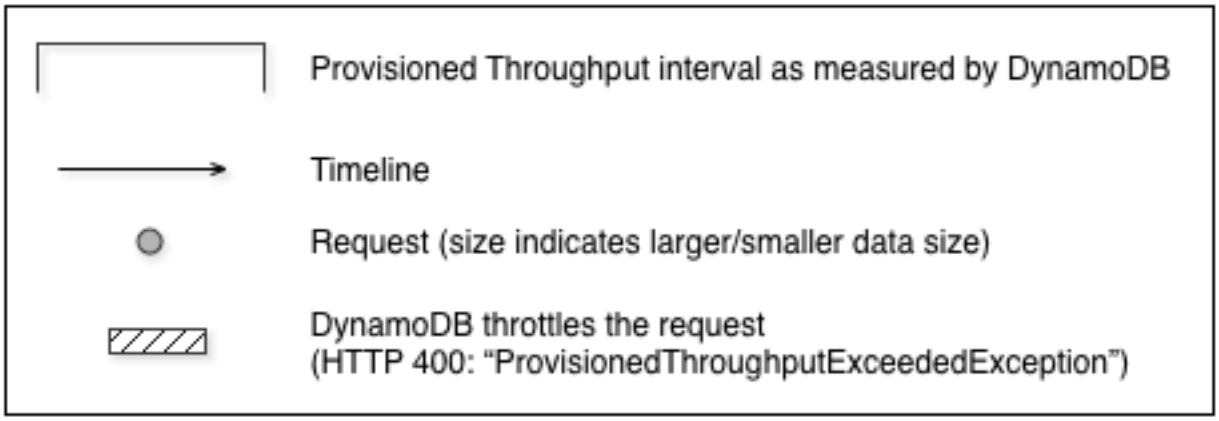
Evitar los picos repentinos en la actividad de lectura

Al crear una tabla, establece los requisitos de unidades de capacidad de lectura y escritura. Para las lecturas, las unidades de capacidad se expresan como el número de solicitudes de consistencia alta de lectura de datos de 4 KB por segundo. En el caso de lecturas eventualmente consistentes, una unidad de capacidad de lectura es de dos solicitudes de lectura de 4 KB por segundo. Una operación `Scan` realiza lecturas eventualmente consistentes de forma predeterminada y puede devolver hasta 1 MB (una página) de datos. Por lo tanto, una única solicitud de `Scan` puede consumir (tamaño de página de 1 MB /tamaño de elemento 4 KB)/2 (lecturas eventualmente consistentes) = 128 operaciones de lectura. Si desea solicitar lecturas fuertemente consistentes en su lugar, la operación `Scan` consumiría el doble de rendimiento aprovisionado, es decir, 256 operaciones de lectura.

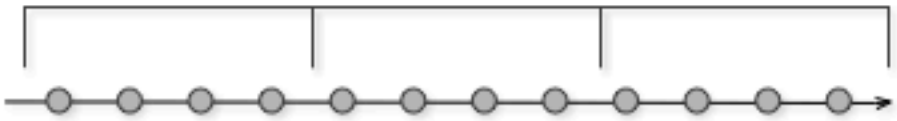
Esto representa un pico repentino de uso en comparación con la capacidad de lectura configurada para la tabla. Este uso de unidades de capacidad en un análisis impide que otras solicitudes más importantes para la misma tabla utilicen las unidades de capacidad disponibles. En consecuencia, es probable que obtenga una excepción `ProvisionedThroughputExceeded` para esas solicitudes.

El problema no reside exclusivamente al aumento repentino de unidades de capacidad que `Scan` utiliza. El análisis probablemente consuma todas sus unidades de capacidad en la misma partición, porque las solicitudes de análisis leen elementos que se encuentran contiguos en la partición. Esto significa que la solicitud se dirige a la misma partición, con lo que provoca que se consuman todas sus unidades de capacidad e impone una limitación controlada que impide que la partición reciba otras solicitudes. Si la solicitud de lectura de datos se distribuye entre varias particiones, la operación no impone una limitación controlada a una partición determinada.

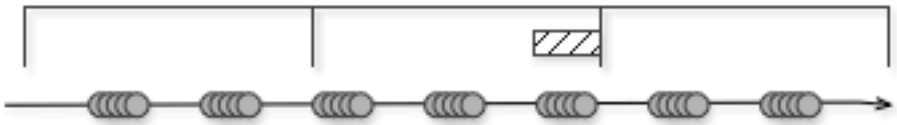
En el siguiente diagrama se ilustra el impacto de un pico repentino del uso de unidades de capacidad que provocan las operaciones `Query` y `Scan`, así como su impacto en las demás solicitudes para la misma tabla.



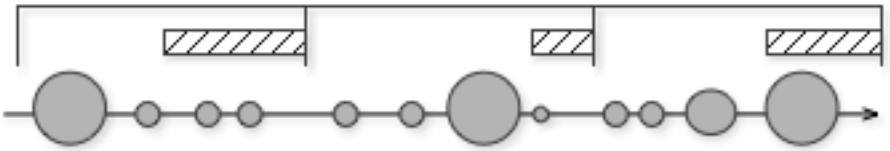
1. Good: Even distribution of requests and size



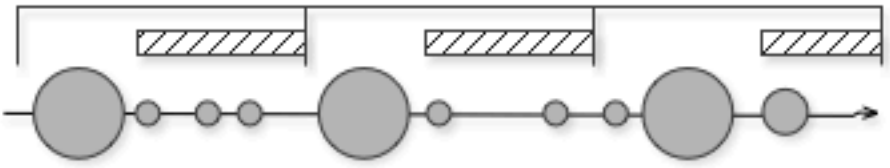
2. Not as Good: Frequent requests in bursts



3. Bad: A few random large requests



4. Bad: Large scan operations



Como se muestra aquí, el pico de uso puede afectar el rendimiento aprovisionado de la tabla de varias maneras:

1. Bueno: distribución uniforme de las solicitudes y el tamaño

2. No tan bueno: solicitudes frecuentes en ráfagas
3. Malo: algunas solicitudes grandes aleatorias
4. Malo: operaciones de análisis grandes

En lugar de utilizar una gran operación `Scan`, puede utilizar las técnicas siguientes para minimizar el impacto de un examen en el desempeño provisionado de una tabla.

- Reducir el tamaño de la página

  Dado que una operación de análisis lee una página completa (de forma predeterminada, 1 MB), puede reducir el impacto de la operación de análisis configurando un tamaño de página menor. La operación `Scan` proporciona el parámetro Limit (Límite) que se puede utilizar para establecer el tamaño de página de la solicitud. Cada solicitud `Query` o `Scan` que tiene un tamaño de página menor utiliza menos operaciones de lectura y crea una "pausa" entre ellas. Por ejemplo, suponga que cada elemento es de 4 KB y establece el tamaño de página en 40 elementos. Una solicitud `Query` consumiría solo 20 operaciones de lectura eventualmente consistentes o 40 operaciones de lectura fuertemente consistentes. Un mayor número de operaciones `Query` o `Scan` permitiría que las demás solicitudes esenciales se realizasen correctamente sin que se aplicara la limitación controlada.

- Aislar las operaciones de análisis

  DynamoDB se ha diseñado para facilitar la escalabilidad. En consecuencia, una aplicación puede crear tablas para fines distintos e, incluso, duplicar contenido entre varias de ellas. Si desea realizar análisis en una tabla que no recibe tráfico "esencial". Algunas aplicaciones controlan esta carga rotando el tráfico cada hora entre dos tablas; una para el tráfico esencial y otra de registro. Las demás aplicaciones pueden efectuar este control realizando cada escritura en dos tablas: una tabla "esencial" y otra tabla "duplicada".

Configure la aplicación de modo que repita las solicitudes que reciban un código de respuesta que indique que se ha superado el rendimiento aprovisionado. O bien, aumente el rendimiento aprovisionado para su tabla mediante la operación `UpdateTable`. Si la carga de trabajo presenta picos temporales que pueden provocar que el rendimiento supere ocasionalmente el nivel aprovisionado, repita la solicitud con retroceso exponencial. Para obtener más información sobre cómo implementar el retardo exponencial, consulte Reintentos de error y retroceso exponencial.

Aprovechamiento de los análisis paralelos

Muchas aplicaciones pueden beneficiarse si se usan las operaciones `Scan` en paralelo, en lugar de análisis secuenciales. Por ejemplo, una aplicación que procesa una gran tabla de datos históricos puede llevar a cabo un análisis en paralelo mucho más rápidamente que uno secuencial. Varios subprocesos de trabajo contenidos en un proceso de "barrido" en segundo plano podrían analizar una tabla con menor prioridad sin afectar al tráfico de producción. En todos estos ejemplos, se utiliza una operación `Scan` de tal forma que no priva a las demás aplicaciones de los recursos de desempeño provisionado.

Aunque los análisis en paralelo pueden ser beneficiosos, pueden suponer una demanda excesiva del rendimiento aprovisionado. Con un análisis paralelo, la aplicación tiene varios trabajadores que están ejecutando operaciones `Scan` simultáneas. Esto puede consumir rápidamente toda la capacidad de lectura aprovisionada de su tabla. En ese caso, las demás aplicaciones que necesiten obtener acceso a la tabla podrían sufrir una limitación controlada que se lo impida.

Un examen en paralelo puede ser la elección adecuada si se cumplen las siguientes condiciones:

- El tamaño de la tabla es de 20 GB o mayor.
- El rendimiento de lectura aprovisionado de la tabla no se está utilizando en su totalidad.
- Las operaciones `Scan` secuenciales son demasiado lentas.

Elección de TotalSegments

La configuración más idónea de `TotalSegments` depende de los datos concretos, de los ajustes de rendimiento aprovisionado de la tabla y de los requisitos de rendimiento. Probablemente deba experimentar para lograr los resultados deseados. Recomendamos comenzar por una proporción sencilla; por ejemplo, un segmento por cada 2 GB de datos. Por ejemplo, en el caso de una tabla de 30 GB, podría establecer `TotalSegments` en 15 (30 GB/2 GB). En este caso, la aplicación utilizaría 15 procesos de trabajo, cada uno de los cuales analizaría un segmento diferente.

También puede elegir un valor de `TotalSegments` que se base en los recursos del cliente. Puede establecer `TotalSegments` en cualquier número comprendido entre 1 y 1 000 000. DynamoDB le permitirá analizar esa cantidad de segmentos. Por ejemplo, si el cliente limita el número de subprocesos que se pueden ejecutar de forma simultánea, puede aumentar gradualmente el valor de `TotalSegments` hasta obtener el mejor rendimiento de `Scan` con la aplicación.

Monitoree los exámenes en paralelo para optimizar la utilización del rendimiento aprovisionado, además de asegurarse de no privar de recursos a las demás aplicaciones. Aumente el valor de `TotalSegments` si no se consume todo el rendimiento aprovisionado pero las solicitudes `Scan` siguen siendo objeto de una limitación controlada. Reduzca el valor de `TotalSegments` si las solicitudes `Scan` consumen más desempeño provisionado del que desea utilizar.

Prácticas recomendadas para el diseño de tablas de DynamoDB

Los principios generales de diseño de Amazon DynamoDB recomiendan utilizar la menor cantidad de tablas posible. En la mayoría de los casos, le recomendamos que considere la posibilidad de utilizar una sola tabla. No obstante, si no es viable utilizar una sola tabla o un número reducido de ellas, estas directrices pueden ser de utilidad.

- El límite por cuenta no puede ser superior a 10 000 tablas por cada cuenta. Si su aplicación requiere más tablas, planifique la distribución de las tablas entre varias cuentas. Para obtener más información, consulte Cuotas de tabla, servicio y cuenta en Amazon DynamoDB.
- Tenga en cuenta los límites de plano de control para las operaciones de plano de control simultáneas que puedan afectar a la administración de las tablas.
- Colabore con los arquitectos de soluciones de AWS para validar sus patrones en diseños de varios inquilinos.

Uso de tablas globales de DynamoDB

Las tablas globales se crean en la huella global de Amazon DynamoDB para proporcionarle una base de datos totalmente administrada, multirregión y multiactiva que puede ofrecer un rendimiento rápido y local, de lectura y de escritura, para aplicaciones globales de escalado masivo. Las tablas globales replican automáticamente las tablas de Amazon DynamoDB en las Regiones de AWS de su elección. No es necesario realizar cambios en la aplicación porque las tablas globales utilizan las API de DynamoDB existentes. No hay costos iniciales ni compromisos por utilizar las tablas globales y solo pagará por los recursos que utilice.

En esta guía se explica cómo utilizar las tablas globales de DynamoDB de forma eficaz. Se proporcionan datos clave sobre las tablas globales, se explican los principales casos de uso de la característica, se describen los dos modos de coherencia, se presenta una taxonomía de tres modelos de escritura diferentes que debe tener en cuenta, se analizan las cuatro opciones principales de enrutamiento de solicitudes que podría implementar, se analizan las formas de

evacuar una región activa o una región sin conexión, se explica cómo pensar en la planificación de la capacidad de rendimiento y se proporciona una lista de aspectos que debe tener en cuenta al implementar las tablas globales.

Esta guía se ajusta a un contexto más amplio de las implementaciones multirregión de AWS, tal como se describe en el documento técnico AWS Multi-Region Fundamentals y en el video Data resiliency design patterns with AWS.

Temas

- Datos clave sobre el diseño de tablas globales de DynamoDB
- Datos clave acerca de la MREC
- Datos clave acerca de la MRSC
- Casos de uso de tablas globales de DynamoDB de MREC
- Modos de escritura con tablas globales de DynamoDB
- Estrategias de enrutamiento de DynamoDB
- Procesos de evacuación
- Planificación de la capacidad de rendimiento para tablas globales de DynamoDB
- Lista de comprobación de preparación para tablas globales de DynamoDB
- Conclusión y recursos

Datos clave sobre el diseño de tablas globales de DynamoDB

- Existen dos versiones de las tablas globales: la versión actual versión 2019.11.21 (actual) de las tablas globales (a veces llamada V2) y Versión 2017.11.29 (heredada) de las tablas globales (a veces llamada V1). Esta guía se centra exclusivamente en la versión actual.
- DynamoDB (sin tablas globales) es un servicio regional, lo que significa que tiene una alta disponibilidad y es intrínsecamente resistente a los errores de infraestructura, tales como el error de toda una zona de disponibilidad. Una tabla de DynamoDB de una sola región está diseñada para tener una disponibilidad del 99,99 %. Para más información, consulte el acuerdo de nivel de servicio (SLA) de DynamoDB.
- Una tabla global de DynamoDB replica sus datos entre dos o más regiones. Una tabla de DynamoDB de varias regiones está diseñada para tener una disponibilidad del 99,999 %. Con una planificación adecuada, las tablas globales pueden contribuir a crear una arquitectura que sea resistente a los errores regionales.

- DynamoDB no dispone de un punto de conexión global. Todas las solicitudes se realizan a un punto de conexión regional que accede a la instancia de tabla global local de esa región.
- Las llamadas a DynamoDB no deben hacerse entre regiones. Se recomienda que una aplicación que se encuentre en una región acceda de manera directa solo al punto de conexión local de DynamoDB correspondiente a su región. Si se detectan problemas en una región (en la capa de DynamoDB o en la pila circundante), el tráfico de los usuarios finales debe redirigirse a un punto de conexión de aplicación distinto alojado en otra región. Las tablas globales garantizan que la aplicación alojada en todas las regiones tenga acceso a los mismos datos.

Modos de coherencia

Al crear una tabla global, se configura el modo de coherencia. Las tablas globales admiten dos modos de coherencia: coherencia final multirregional (MREC) y coherencia alta multirregional (MRSC) que se ingresó en junio de 2025.

Si no especifica un modo de coherencia al crear una tabla global, esta adoptará de manera predeterminada la MREC. Una tabla global no puede contener réplicas configuradas con distintos modos de coherencia. No puede cambiar el modo de coherencia de una tabla global después de su creación.

Datos clave acerca de la MREC

- Las tablas globales que utilizan la MRSC emplean un modelo de replicación activa-activa. Desde la perspectiva de DynamoDB, la tabla en cada región tiene la misma capacidad para aceptar solicitudes de lectura y escritura. Después de recibir una solicitud de escritura, la tabla de réplica local replica la operación de escritura en otras regiones remotas participantes en segundo plano.
- Los elementos se replican de forma individual. Es posible que los elementos que se actualizan en una misma transacción no puedan replicarse juntos.
- Cada partición de tabla de la región de origen replica sus operaciones de escritura en paralelo con las demás particiones. La secuencia de las operaciones de escritura en la región remota podría no coincidir con la secuencia de las operaciones de escritura que se produjo en la región de origen. Para obtener más información sobre las particiones de tablas, consulte la entrada de blog Escalado de DynamoDB: cómo afectan al rendimiento las particiones, las claves activas y la división por actividad.
- Un elemento que se acaba de escribir se propagará normalmente a todas las réplicas de tabla en cuestión de segundos. Las regiones cercanas tienden a propagarse más rápido.

- Amazon CloudWatch proporciona una métrica `ReplicationLatency` para cada par de regiones. Se calcula mediante la observación de los elementos que llegan, la comparación de su tiempo de llegada con su tiempo de escritura inicial y el cálculo de un promedio. Los tiempos se almacenan en CloudWatch en la región de origen. La visualización de los tiempos medio y máximo puede resultar útil para determinar el retraso medio y en el peor de los casos de la réplica. No hay SLA para esta latencia.
- Si un elemento individual se actualiza más o menos al mismo tiempo (en este intervalo de `ReplicationLatency`) en dos regiones distintas y la segunda operación de escritura se produce antes de que se replicara la primera, existe la posibilidad de que se produzcan conflictos de escritura. Las tablas globales que utilizan la MREC resuelven estos conflictos con un mecanismo del último escritor gana, basado en la marca de tiempo de las operaciones de escritura. La primera operación “pierde” frente a la segunda operación. Estos conflictos no se registran en CloudWatch ni en AWS CloudTrail.
- Cada elemento tiene una marca de tiempo de última escritura que se mantiene como una propiedad de sistema privada. El enfoque del último escritor gana se implementa mediante una operación de escritura condicional que requiere que la marca de tiempo del elemento entrante sea mayor que la marca de tiempo del elemento existente.
- Una tabla global replica todos los elementos en todas las regiones participantes. Si quiere tener distintos ámbitos de replicación, puede crear varias tablas globales y asignar a cada una distintas regiones participantes.
- La región local acepta las operaciones de escritura aunque la región de réplica no tenga conexión o crezca `ReplicationLatency`. La tabla local sigue intentando replicar elementos en la tabla remota hasta que cada elemento lo consigue.
- En el improbable caso de que una región esté totalmente sin conexión, cuando vuelva a conectarse más tarde, se volverán a intentar todas las réplicas salientes y entrantes pendientes. No es necesaria ninguna acción especial para volver a sincronizar las tablas. El mecanismo del último escritor gana garantiza que, al final, los datos pasan a ser coherentes.
- En cualquier momento, puede agregar una región nueva a una tabla de la MREC de DynamoDB. DynamoDB gestiona la sincronización inicial y la replicación continua. También puede eliminar una región (incluso la región original), y esto eliminará la tabla local de esa región.

Datos clave acerca de la MRSC

- Las tablas globales que utilizan la MRSC emplean un modelo de replicación activa-activa. Desde la perspectiva de DynamoDB, la tabla en cada región tiene la misma capacidad para aceptar

solicitudes de lectura y escritura. Los cambios de elementos en la réplica de una tabla global de MRSC se replican de manera sincrónica en al menos otra región antes de que la operación de escritura devuelva una respuesta correcta.

- Las operaciones de lectura altamente coherente en cualquier réplica de MRSC siempre devuelven la versión más reciente de un elemento. Las operaciones de escritura condicionales siempre evalúan la expresión de la condición, ya que la comparan con la versión más reciente de un elemento. Las actualizaciones siempre se llevan a cabo con la última versión de un elemento.
- Es posible que las operaciones de lectura coherente posterior en una réplica de la MRSC no incluyan los cambios que apenas se hayan producido en otra región y que ni siquiera incluyan los cambios que se produjeron hace muy poco tiempo en la misma región.
- Una operación de escritura produce un error con una excepción `ReplicatedWriteConflictException` cuando intenta modificar un elemento que ya se está modificando en otra región. Las operaciones de escritura en las que se produce un error con la excepción `ReplicatedWriteConflictException` se pueden volver a intentar y se ejecutarán correctamente si el elemento ya no se está modificando en otra región.
- Con la MRSC, las latencias son más altas para las operaciones de escritura y para las operaciones de lectura altamente coherente. Es necesaria una comunicación entre regiones para estas operaciones. Esta comunicación puede agregar latencia, que aumenta en función de la latencia de ida y vuelta entre la región a la que se accede y la región más cercana que participa en la tabla global. Para más información, consulte la presentación de AWS re:Invent 2024, Multi-Region strong consistency with Amazon DynamoDB global tables. Las operaciones de lectura coherente posterior no tienen latencia adicional. Existe una herramienta de prueba de código abierto que le permite calcular estas latencias de manera experimental con sus regiones.
- Los elementos se replican de forma individual. Las tablas globales que utilizan la MRSC no admiten las API de transacciones.
- Una tabla global de MRSC se debe implementar exactamente en tres regiones. Puede configurar una tabla global de MRSC con tres réplicas o con dos réplicas y un testigo. Un testigo es un componente de una tabla global de la MRSC que contiene los datos recientes escritos en las réplicas de las tablas globales. Un testigo proporciona una alternativa opcional a una réplica completa, además de que admite la arquitectura de disponibilidad de la MRSC. No puede realizar operaciones de lectura o escritura en un testigo. Un testigo no genera costos de almacenamiento ni escritura. Un testigo se encuentra en una región distinta de las dos réplicas.
- Para crear una tabla global de MRSC, agregue una réplica y un testigo o dos réplicas a una tabla de DynamoDB existente que no contenga datos. No puede agregar réplicas adicionales a una tabla global de MRSC existente. No puede eliminar una réplica única o un testigo de una tabla

global de la MRSC. Puede eliminar dos réplicas o eliminar una réplica y un testigo de una tabla global de la MRSC. En el segundo escenario se convierte la réplica restante en una tabla de DynamoDB de una sola región.

- Puede determinar si una tabla global de la MRSC tiene un testigo configurado, y en qué región está configurado, desde la salida de la API DescribeTable. El testigo es responsabilidad de DynamoDB y este lo administra. No aparece en la Cuenta de AWS de la región en la que esté configurado.
- Las tablas globales de MRSC están disponibles en los siguientes conjuntos de regiones:
  - Conjunto de regiones de EE. UU.: Este de EE. UU. (Norte de Virginia), Este de EE. UU. (Ohio), Oeste de EE. UU. (Oregón)
  - Conjunto de regiones de la Unión Europea: Europa (Irlanda), Europa (Londres), Europa (París), Europa (Fráncfort)
  - Conjunto de regiones de Asia-Pacífico: Asia-Pacífico (Tokio), Asia-Pacífico (Seúl) y Asia-Pacífico (Osaka).
- Las tablas globales de la MRSC no pueden abarcar conjuntos de regiones. Por ejemplo, una tabla global de la MRSC no puede contener réplicas de conjuntos de regiones de EE. UU. y de la UE.
- El Tiempo de vida (TTL) no se admite para las tablas globales de MRSC.
- Los índices secundarios locales (LSI) no se admiten para las tablas globales MRSC.
- La información de CloudWatch Contributor Insights solo se comunica para la región en la que se ha producido una operación.
- La región local acepta todas las operaciones de lectura y escritura siempre y cuando haya una segunda región que aloje una réplica o un testigo para establecer el cuórum. Si una segunda región no está disponible, la región local solo podrá ofrecer operaciones de lectura coherente posterior.
- En el caso improbable de que una región quede totalmente sin conexión, cuando vuelva a conectarse, se actualizará de manera automática. Hasta que se ponga al día, las operaciones de escritura y las operaciones de lectura altamente coherente solo en la región que se está poniendo al día devolverán errores, mientras que las solicitudes a otras regiones seguirán funcionando con normalidad. Las operaciones de lectura coherente posterior en la región que se está poniendo al día devolverán los datos que hasta ahora se propagaron a la región, con un comportamiento de coherencia local habitual entre el nodo líder y las réplicas locales. No es necesaria ninguna acción especial para volver a sincronizar las tablas.

Casos de uso de tablas globales de DynamoDB de MREC

Las tablas globales de MREC ofrecen los beneficios siguientes:

- Operaciones de lectura de baja latencia. Coloque una copia de los datos más cerca del usuario final para reducir la latencia de red durante las operaciones de lectura. Los datos se mantienen tan actualizados como el valor de `ReplicationLatency`.
- Operaciones de escritura de baja latencia. Puede escribir en una región cercana para reducir la latencia de red y el tiempo necesario para realizar la escritura. El tráfico de escritura debe enrutarse cuidadosamente para garantizar que no haya conflictos. Las técnicas de enrutamiento se describen más detalladamente en Estrategias de enrutamiento de DynamoDB.
- Migración de región sin problemas. Puede agregar una nueva región y eliminar la antigua para migrar una implementación de una región en otra, todo ello sin tiempo de inactividad en la capa de datos.

Las tablas globales de MREC y MRSC ofrecen este beneficio:

- Mayor resiliencia y recuperación de desastres. Si el rendimiento de una región ha empeorado o se ha producido una interrupción total, puede evacuarla. Evacuar significa retirar algunas o todas las solicitudes que se dirijan a esa región. El uso de tablas globales aumenta el SLA de DynamoDB respecto del porcentaje de tiempo de actividad mensual del 99,99 % al 99,999 %. El uso de la MREC admite un objetivo de punto de recuperación (RPO) y un objetivo de tiempo de recuperación (RTO) medidos en segundos. El uso de la MRSC admite un RPO igual a cero.

  Por ejemplo, Fidelity Investments presentó en re:Invent 2022 cómo utilizan las tablas globales de DynamoDB para su sistema de administración de pedidos. Su objetivo era lograr un procesamiento con baja latencia de manera fiable a una escala que no podrían alcanzar con el procesamiento en las instalaciones, además de que mantenían la resiliencia ante errores de zona de disponibilidad y de región.

Si el objetivo es la resiliencia y la recuperación ante desastres, las tablas de la MRSC tienen latencias de escritura más altas y latencias de lectura altamente coherentes y más altas, pero admiten un RPO igual a cero. Las tablas globales de MREC admiten un RPO igual al retraso de la replicación entre réplicas, que suele ser de unos pocos segundos, en función de las regiones de las réplicas.

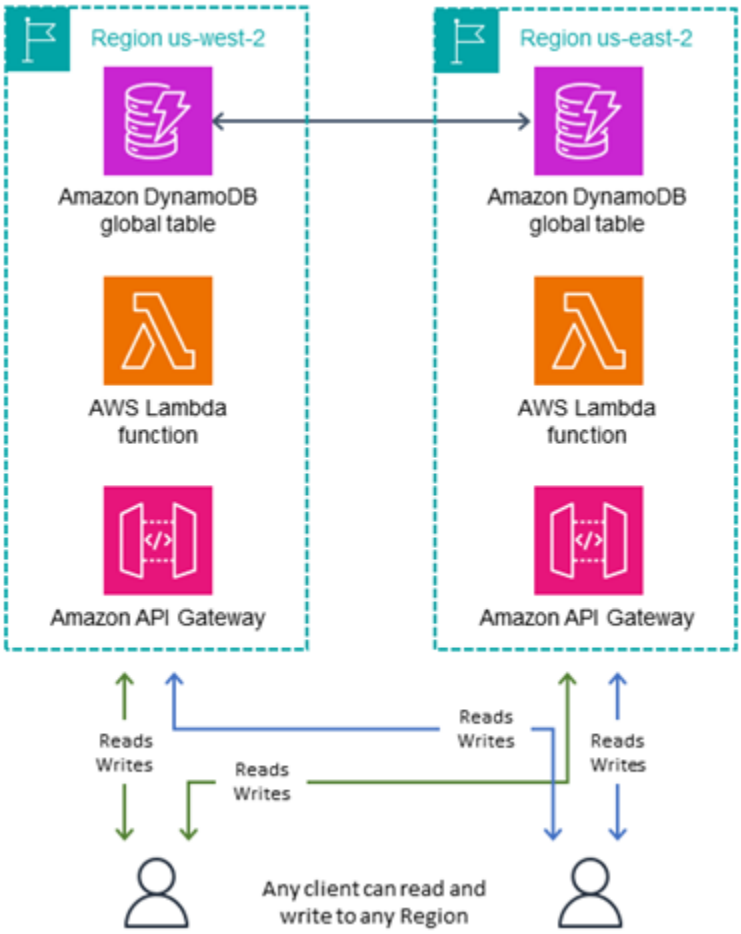
Modos de escritura con tablas globales de DynamoDB

Las tablas globales son siempre activas-activas en el nivel de tabla. No obstante, en especial para las tablas de la MREC, es conveniente tratarlas como activas-pasivas mediante el control del enrutamiento de las solicitudes de escritura. Por ejemplo, podría decidir enrutar las solicitudes de escritura a una sola región para evitar posibles conflictos de escritura que puedan producirse con las tablas de MREC.

Hay tres patrones principales de escritura administrada, como se explica en las tres secciones siguientes. Debe considerar qué patrón de escritura se ajusta a su caso de uso. Esta elección afecta a la forma de enrutar las solicitudes, evacuar una región y gestionar la recuperación de desastres. La guía de las secciones posteriores depende del modo de escritura de la aplicación.

Modo de escritura en cualquier región (no principal)

El modo de escritura en cualquier región (que se ilustra en el diagrama siguiente) es totalmente activo-activo y no impone restricciones sobre dónde puede producirse una escritura. Cualquier región puede aceptar una escritura en cualquier momento. Este es el modo más sencillo. Pero solo se puede utilizar con algunos tipos de aplicaciones. Este modo es adecuado para todas las tablas de la MRSC. Resulta también adecuado para las tablas de MREC cuando todos los escritores son idempotentes y, por lo tanto, repetibles de forma segura para que las operaciones de escritura simultáneas o repetidas en las distintas regiones no entren en conflicto. Por ejemplo, cuando un usuario actualiza sus datos de contacto. Este modo también sirve para un caso especial de ser idempotente, un conjunto de datos de solo adición en el que todas las escrituras son inserciones únicas con una clave principal determinista. Por último, este modo resulta adecuado para MREC cuando el riesgo de escrituras en conflicto sea aceptable.



El modo de escritura en cualquier región es la arquitectura más sencilla de implementar. El enrutamiento es más sencillo porque cualquier región puede ser el destino de escritura en cualquier momento. La conmutación por error es más fácil, porque con las tablas de la MRSC, los elementos siempre están sincronizados y, con las tablas de la MRSC, las escrituras recientes se pueden reproducir cualquier número de veces en cualquier región secundaria. Siempre que sea posible, debe efectuar el diseño para este modo de escritura.

Por ejemplo, varios servicios de transmisión de video utilizan tablas globales para hacer el seguimiento de marcadores, reseñas, marcas de estado de observación, etc. Estas implementaciones utilizan tablas de MREC porque necesitan réplicas repartidas por todo el mundo. Cada réplica proporciona operaciones de lectura y escritura de baja latencia. Estas implementaciones pueden utilizar el modo de escritura en cualquier región siempre y cuando que garanticen que todas las operaciones de escritura sean idempotentes. Esto ocurrirá si todas las actualizaciones (por ejemplo, establecer un código nuevo de última hora, asignar una revisión nueva o establecer un estado de observación nuevo) asignan de manera directa el nuevo estado del usuario y el siguiente valor correcto de un elemento no depende de su valor actual. Si, por

casualidad, las solicitudes de escritura del usuario se enrutan a regiones distintas, la última operación de escritura persistirá y el estado global se establecerá de acuerdo con la última asignación. Las operaciones de lectura en este modo acabarán siendo coherentes, retrasadas según el último valor de `ReplicationLatency`.

En otro ejemplo, una empresa de servicios financieros utiliza tablas globales como parte de un sistema para mantener un recuento continuo de las compras con tarjeta de débito de cada cliente, con el fin de calcular las recompensas en efectivo de ese cliente. Quieren conservar un elemento `RunningBalance` por cliente. Por naturaleza, este modo de escritura no es idempotente porque, a medida que llegan las transacciones, modifican el saldo mediante una expresión `ADD` en la que el nuevo valor correcto depende del valor actual. Al utilizar las tablas de la MRSC, pueden seguir escribiendo en cualquier región, ya que cada llamada de `ADD` siempre funciona con el valor más reciente del elemento.

Un tercer ejemplo es el de una empresa que ofrece servicios de publicación de anuncios en línea. Esta empresa decidió que sería aceptable un bajo riesgo de pérdida de datos para conseguir las simplificaciones de diseño del modo de escritura en cualquier región. Cuando publican anuncios, solo disponen de unos milisegundos para recuperar los metadatos suficientes para determinar qué anuncio mostrar y, a continuación, registrar la impresión del anuncio para que no se repita el mismo anuncio en breve. Utilizan tablas globales para obtener operaciones de lectura de baja latencia para los usuarios finales de todo el mundo y operaciones de escritura de baja latencia. Registran todas las impresiones de anuncios de un usuario en un solo elemento, que se representa como una lista creciente. Utilizan un elemento en lugar de agregarlo a una colección de elementos, por lo que pueden eliminar impresiones de anuncios anteriores como parte de cada operación de escritura sin tener que pagar por una operación de eliminación. Esta operación de escritura no es idempotente. Si el mismo usuario final ve anuncios publicados desde varias regiones casi al mismo tiempo, existe la posibilidad de que una operación de escritura para una impresión de anuncios sobrescriba otra. El riesgo es que un usuario podría ver un anuncio repetido de vez en cuando. Decidieron que es aceptable.

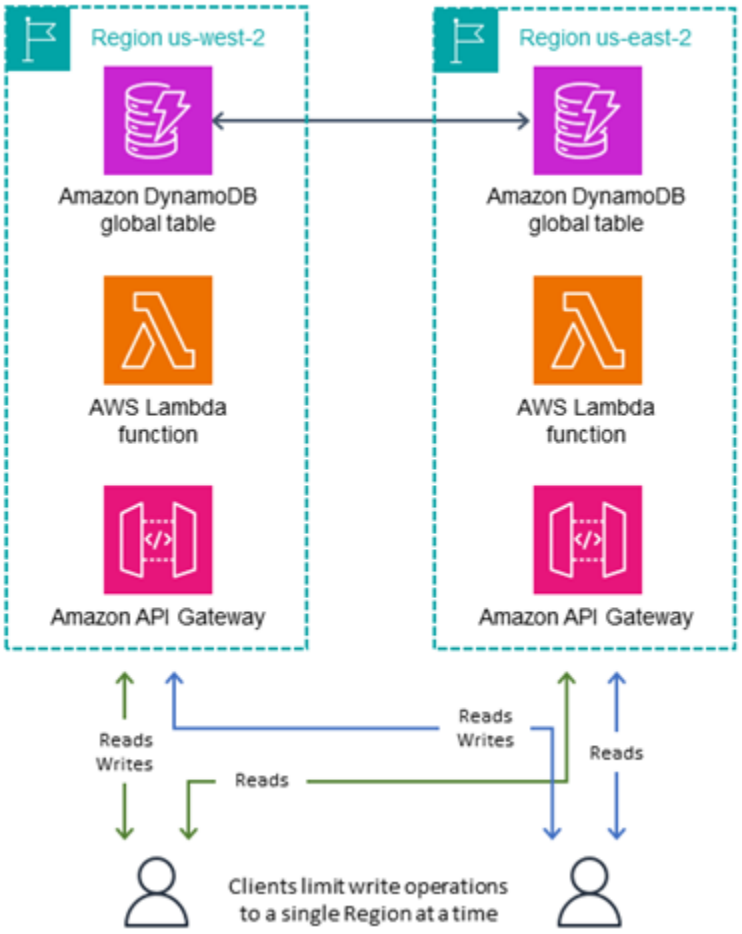
Escritura en una región (única principal)

El modo de escritura en una región, que se muestra en el diagrama siguiente, es activo-pasivo y enruta todas las escrituras de tablas a una sola región activa. Tenga en cuenta que DynamoDB no tiene la noción de una única región activa; el enrutamiento de la aplicación fuera de DynamoDB administra esta situación. El modo de escritura en una región funciona bien para las tablas de MREC que deben evitar conflictos de escritura, ya que garantiza que las operaciones de escritura fluyan solo a una región a la vez. Este modo de escritura es útil cuando quiere utilizar expresiones

condicionales y no puede utilizar la MRSC por algún motivo, o cuando debe hacer transacciones. Estas expresiones no son posibles a menos que se sepa que se están procesando los datos más recientes, por lo que es necesario enviar todas las solicitudes de escritura a una sola región que tenga los datos más recientes.

Cuando utilice una tabla de MRSC, puede optar por escribir en general en una región para mayor practicidad. Por ejemplo, esto puede ayudar a minimizar la ampliación de la infraestructura más allá de DynamoDB. El modo de escritura seguiría siendo escribir en cualquier región, ya que con la MRSC se puede escribir de manera segura en cualquier región en cualquier momento sin preocuparse por la resolución de conflictos que haría que las tablas de la MREC eligieran escribir en una región.

Las lecturas coherentes posteriores pueden ir a cualquier región de réplica para obtener latencias más bajas. Las lecturas altamente coherentes deben ir a la única región principal.



A veces, es necesario cambiar la región activa en respuesta a un error regional. Algunos usuarios cambian la región activa en ese momento de manera periódica, por ejemplo, como una

implementación del tipo seguir el sol. Esto sitúa la región activa cerca de la zona geográfica con mayor actividad (suele ser donde es de día, de ahí el nombre), lo que supone una latencia mínima en las operaciones de lectura y escritura. También tiene el otro beneficio de llamar al código de cambio de región a diario y garantizar que se haya probado correctamente antes de que haya que realizar una recuperación ante desastres.

Las regiones pasivas pueden mantener un conjunto reducido de infraestructura en torno a DynamoDB que se crea solo si se convierte en la región activa. En esta guía no se abordan los diseños de llama piloto y espera activa. Para obtener más información, consulte Disaster Recovery (DR) Architecture on AWS, Part III: Pilot Light and Warm Standby.

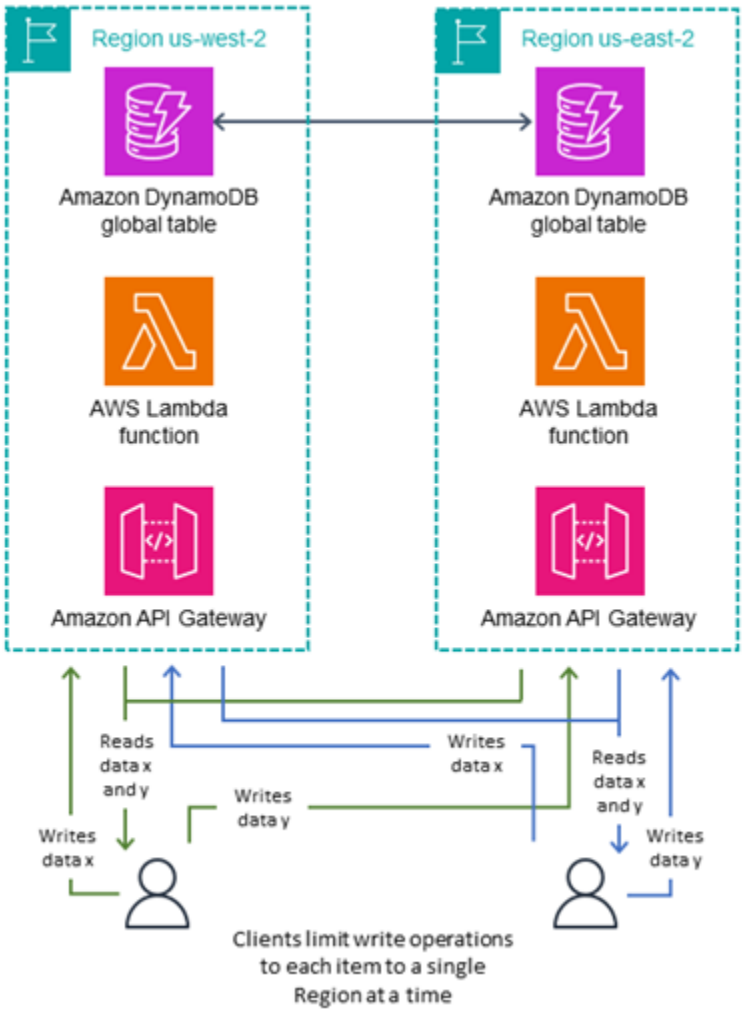
El modo de escritura en una región funciona bien cuando se utilizan tablas globales para operaciones de lectura distribuidas de manera global y con baja latencia. Un ejemplo es una empresa grande de redes sociales que necesita tener los mismos datos de referencia disponibles en todas las regiones del mundo. No actualizan los datos con frecuencia, pero cuando los actualizan, escriben solo en una región para evitar posibles conflictos de escritura. Las operaciones de lectura siempre se permiten desde cualquier región.

Tomemos como otro ejemplo la empresa de servicios financieros mencionada anteriormente que implementó el cálculo diario de devoluciones en efectivo. Utilizaban el modo de escritura en cualquier región para calcular el saldo, pero utilizaban el modo de escritura en una región para hacer el seguimiento de los pagos. Para ello hay que hacer transacciones, que no son compatibles con las tablas de la MRSC, así que funciona mejor con una tabla de la MREC independiente y el modo de escritura a una región.

Escritura en la región (principal mixta)

El modo de escritura en su región, que se ilustra en el diagrama siguiente, funciona con las tablas de la MREC. Asigna distintos subconjuntos de datos a distintas regiones de origen y permite hacer operaciones de escritura en un elemento solo a través de su región de origen. Este modo es activo-pasivo, pero asigna la región activa según el elemento. Cada región es la principal para su propio conjunto de datos no superpuesto. Las operaciones de escritura deben protegerse para garantizar una ubicación adecuada.

Este modo es similar a escribir en una región, salvo que permite operaciones de escritura con menor latencia, ya que los datos asociados a cada usuario pueden colocarse en una red más cercana a dicho usuario. También distribuye la infraestructura circundante de manera más uniforme entre las regiones y es necesario menos trabajo para crear la infraestructura durante un escenario de conmutación por error, porque todas las regiones tienen una parte de su infraestructura ya activa.



Puede determinar la región de origen de los elementos de varias maneras:

- Tipo intrínseco: algún aspecto de los datos, como un atributo especial o un valor incrustado en su clave de partición, deja clara su región de origen. Esta técnica se describe en la entrada de blog Use Region pinning to set a home Region for items in an Amazon DynamoDB global table.
- Negociado: la región de origen de cada conjunto de datos se negocia de manera externa. Por ejemplo, con un servicio global independiente que mantiene las asignaciones. La asignación puede tener una duración finita, tras la cual está sujeta a renegociación.
- Orientado a tablas: en lugar de crear una única tabla global replicada, se crea el mismo número de tablas globales que regiones replicadas. El nombre de cada tabla indica su región de origen. En las operaciones estándar, todos los datos se escriben en la región de origen, mientras que las demás regiones conservan una copia de solo lectura. Durante una conmutación por error, otra región adopta de manera temporal las tareas de escritura para esa tabla.

Por ejemplo, imagine que trabaja para una empresa de juegos. Necesita operaciones de lectura y escritura de baja latencia para todos los jugadores del mundo. Asigna a cada jugador a la región más cercana a este. Esa región se encarga de todas sus operaciones de lectura y escritura, lo que garantiza una coherencia alta entre las operaciones de lectura y escritura. Sin embargo, cuando un jugador viaja o si su región de origen sufre una interrupción del servicio, hay una copia completa de sus datos disponible en regiones alternativas, y se le puede asignar al jugador una región de origen distinta.

Por poner otro ejemplo, imagine que trabaja en una empresa de videoconferencias. Los metadatos de cada conferencia telefónica se asignan a una región concreta. Quienes llamen pueden utilizar la región más cercana para obtener la latencia más baja. Si se produce una interrupción en una región, el uso de tablas globales permite una recuperación rápida porque el sistema puede trasladar el procesamiento de la llamada a otra región en la que ya exista una copia replicada de los datos.

Resumen

- El modo de escritura en cualquier región es adecuado para las tablas de MRSC y las llamadas idempotentes a las tablas de MREC.
- El modo de escritura en una región es adecuado para las llamadas a las tablas de MREC.
- El modo de escritura en su región es adecuado para las llamadas no idempotentes a las tablas de MREC, donde es importante que los clientes escriban en una región cercana a estos.

Estrategias de enrutamiento de DynamoDB

Quizá la parte más compleja de la implementación de una tabla global sea administrar el enrutamiento de las solicitudes. Las solicitudes deben pasar primero de un usuario final a una región elegida y enrutada de alguna manera. La solicitud se encuentra con una pila de servicios en esa región, incluida una capa de computación que quizá consista en un equilibrador de carga respaldado por una función de AWS Lambda, un contenedor o un nodo de Amazon Elastic Compute Cloud (Amazon EC2) y posiblemente otros servicios, incluida otra base de datos. Esa capa de computación se comunica con DynamoDB. Debe hacerlo mediante la utilización del punto de conexión local correspondiente a esa Región. Los datos de la tabla global se replican en las demás regiones participantes y cada región dispone de una pila similar de servicios en torno a su tabla de DynamoDB.

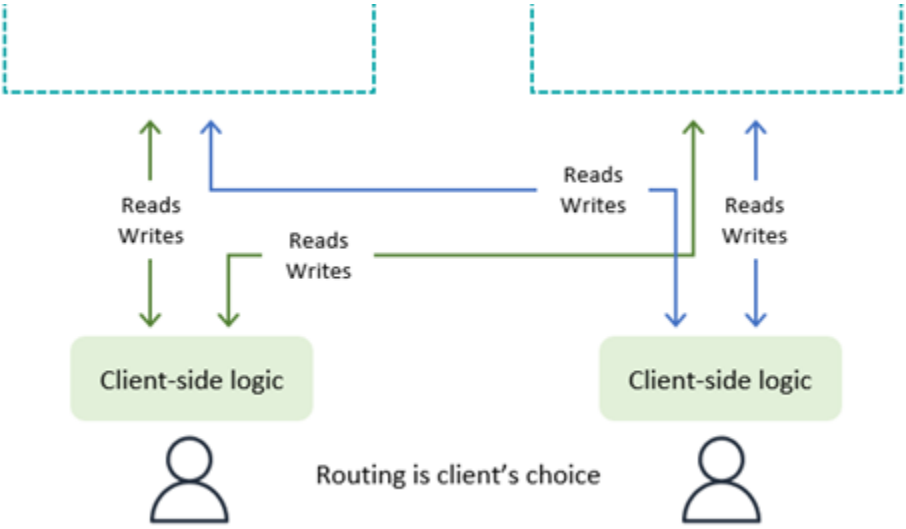
La tabla global proporciona a cada pila de las distintas regiones una copia local de los mismos datos. Podría considerar la posibilidad de diseñar para una única pila en una única Región y prever la

realización de llamadas remotas al punto de conexión de DynamoDB de una Región secundaria si se produce algún problema con la tabla de DynamoDB local. No es una práctica recomendada. Si DynamoDB provoca un problema en una región (o, más probablemente, otra parte de la pila u otro servicio que dependa de DynamoDB), es mejor dirigir al usuario final a otra región para que lo procese y utilice la capa de computación de esa otra región, que se comunicará con su punto de conexión local de DynamoDB. Este enfoque evita por completo la región problemática. Para garantizar la resiliencia, es necesaria la replicación entre varias regiones: la replicación de la capa de computación y la capa de datos.

Existen numerosas técnicas alternativas para enrutar una solicitud de un usuario final a una región para que se procese. La elección óptima depende de su modo de escritura y de sus consideraciones de conmutación por error. En esta sección se analizan cuatro opciones: basada en el cliente, capa de computación, Route 53 y Global Accelerator.

Enrutamiento de solicitudes basado en el cliente

Con el enrutamiento de solicitudes basado en el cliente, que se ilustra en el diagrama siguiente, el cliente del usuario final (una aplicación, una página web con JavaScript u otro cliente) hace un seguimiento de los puntos de conexión de la aplicación válidos (por ejemplo, un punto de conexión de Amazon API Gateway en lugar de un punto de conexión literal de DynamoDB) y utiliza su propia lógica integrada para elegir la región con la que comunicarse. Podría elegir según una selección al azar, de las latencias más bajas observadas, de las mediciones de ancho de banda más altas observadas o de comprobaciones de estado hechas localmente.



Como ventaja, el enrutamiento de solicitudes basado en el cliente es que puede adaptarse a factores como las condiciones reales de tráfico del internet público, para cambiar de región si detecta que el

rendimiento disminuye. El cliente debe conocer todos los posibles puntos de conexión, pero lanzar un nuevo punto de conexión regional no es algo frecuente.

Con el modo de escritura en cualquier región, un cliente puede seleccionar de manera unilateral su punto de conexión preferido. Si su acceso a una región se ve afectado, el cliente puede enrutarse a otro punto de conexión.

Con el modo de escritura en una región, el cliente necesitará un mecanismo para enrutar sus escrituras a la región activa en ese momento. Podría ser algo tan básico como comprobar empíricamente qué región acepta actualmente escrituras (se debe tener en cuenta cualquier rechazo de escritura y conmutar por error a una alternativa) o tan complejo como llamar a un coordinador global para consultar el estado actual de la aplicación (quizás basado en el control de enrutamiento del Controlador de recuperación de aplicaciones [ARC] de Amazon, que proporciona un sistema basado en cuórum de cinco regiones para mantener el estado global para necesidades como esta). El cliente puede decidir si las lecturas pueden ir a cualquier región para obtener una coherencia puntual o si deben dirigirse a la región activa para obtener una coherencia sólida. Para obtener más información, consulte Funcionamiento de Route 53.

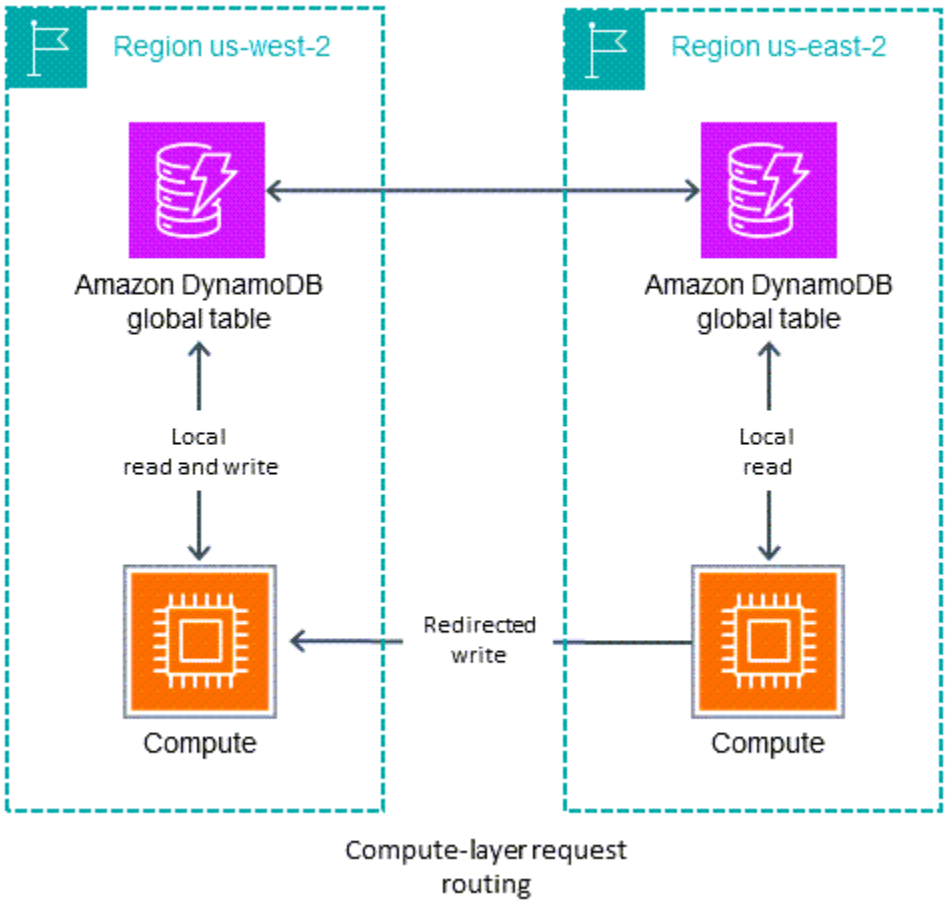
Con el modo de escritura en su región, el cliente necesita determinar la región de origen correspondiente al conjunto de datos con el que trabaja. Por ejemplo, si el cliente corresponde a una cuenta de usuario y cada cuenta de usuario está asignada a una región, el cliente puede solicitar el punto de conexión adecuado a un sistema de inicio de sesión global.

Por ejemplo, una empresa de servicios financieros que ayuda a los usuarios a administrar las finanzas de su empresa a través de la web podría utilizar tablas globales con un modo de escritura en su región. Cada usuario debe iniciar sesión en un servicio central. Ese servicio devuelve las credenciales y el punto de conexión de la región en la que funcionarán esas credenciales. Las credenciales son válidas durante un período breve. Transcurrido ese tiempo, la página web negocia automáticamente un nuevo inicio de sesión, lo que brinda la oportunidad de redirigir potencialmente la actividad del usuario a una nueva región.

Enrutamiento de solicitudes de la capa de computación

Con el enrutamiento de solicitudes de la capa de computación (que se ilustra en el diagrama siguiente), el código que se ejecuta en dicha capa determina si procesar la solicitud de manera local o pasarla a una copia de sí mismo que se esté ejecutando en otra región. Cuando utilice el modo de escritura en una región, la capa de computación podría detectar que no es la región activa y permitir las operaciones de lectura locales mientras reenvía todas las operaciones de escritura a otra región. Este código de la capa de computación debe conocer la topología de los datos y las reglas

de enrutamiento, y aplicarlas de manera fiable según la última configuración que especifique qué regiones están activas para los datos determinados. La pila de software externa en la región no tiene por qué conocer cómo el microservicio enruta las solicitudes de lectura y de escritura. En un diseño sólido, la región receptora valida si es la principal actual para la operación de escritura. Si no lo es, genera un error que indica que es necesario corregir el estado global. La Región receptora también podría almacenar en búfer la operación de escritura durante un tiempo si la región principal está en proceso de cambiar. En todos los casos, la pila de computación de una región escribe solo en su punto de conexión de DynamoDB local, pero las pilas de computación podrían comunicarse entre sí.

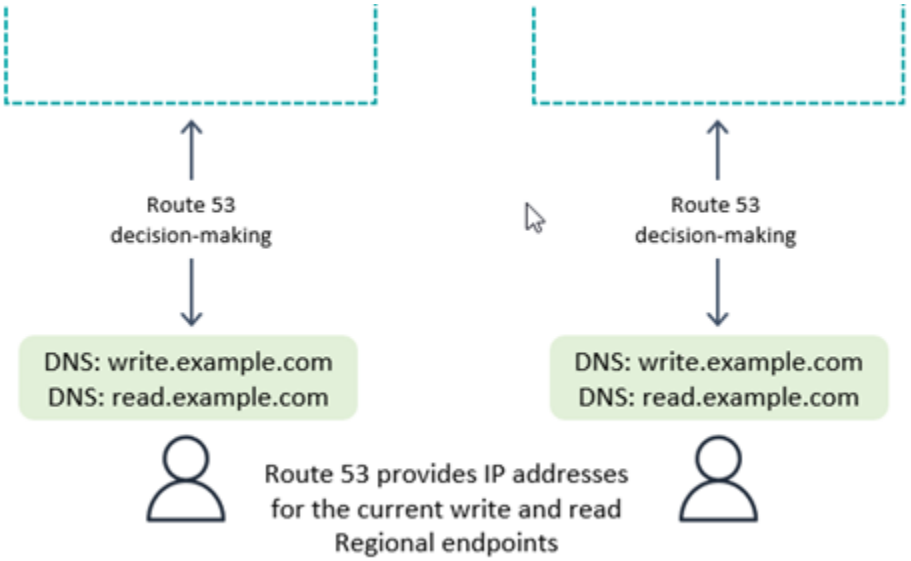


The Vanguard Group utiliza un sistema denominado Herramienta de orquestación y estado global (GOaST) y una biblioteca denominada Biblioteca multirregión global (GMRlib) para este proceso de enrutamiento, tal como se presentó en re:Invent 2022. Utilizan un modelo principal único del tipo “seguir el sol”. La GOaST mantiene el estado global, similar al control de enrutamiento de ARC que se analizó en la sección anterior. Utiliza una tabla global para saber cuál es la región principal y cuándo está prevista la próxima transición de región principal. Todas las operaciones de lectura y escritura pasan por la GMRlib, que se coordina con la GOaST. La GMRlib permite que las operaciones de lectura se hagan de manera local, con baja latencia. En el caso de las operaciones

de escritura, la GMRlib verifica si la región local es la principal actual. Si es así, la operación de escritura se completa directamente. De lo contrario, la GMRlib reenvía la tarea de escritura a la GMRlib de la región principal. La biblioteca receptora confirma que también se considera la región principal y, si no lo es, genera un error, lo que indica un retraso de propagación con el estado global. Este enfoque proporciona un beneficio de validación al no escribir directamente en un punto de conexión de DynamoDB remoto.

Enrutamiento de solicitudes de Route 53

El Controlador de recuperación de aplicaciones (ARC) de Amazon es una tecnología de servicio de nombres de dominio (DNS). Con Route 53, el cliente solicita su punto de conexión mediante la búsqueda de un nombre de dominio DNS conocido y Route 53 le devuelve la dirección IP correspondiente a los puntos de conexión regionales que considere más adecuados. Esto se explica en el siguiente diagrama. Route 53 tiene una lista larga de políticas de enrutamiento que utiliza para determinar la región adecuada. También puede hacer un enrutamiento de conmutación por error para desviar el tráfico de las regiones que no superen las comprobaciones de estado.



Con el modo de escritura en cualquier región, o si se combina con el enrutamiento de solicitudes de la capa de computación en el backend, Route 53 puede tener pleno acceso para devolver la región a partir de cualquier regla interna compleja, como la región más próxima a la red, la más próxima geográficamente o cualquier otra opción.

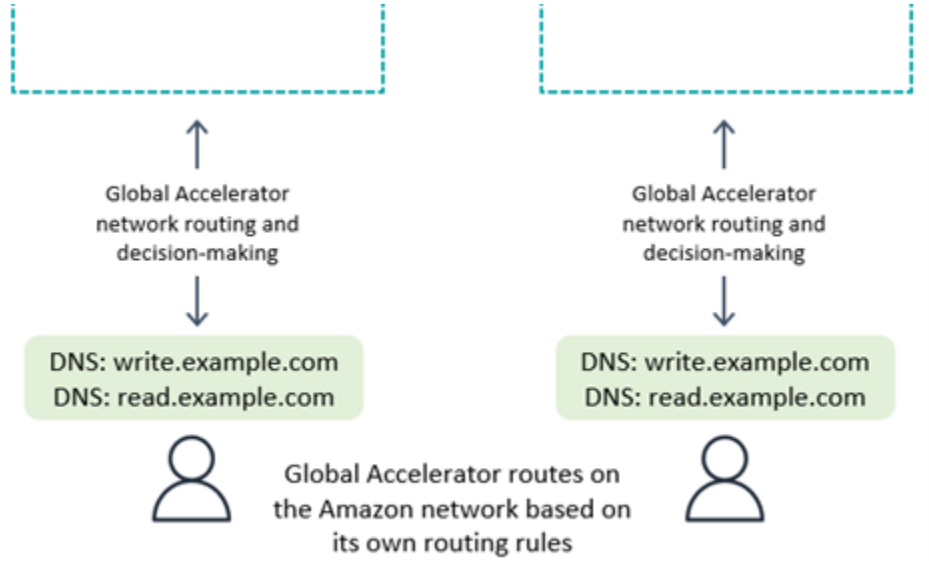
Con el modo de escritura en una región, puede configurar Route 53 para que devuelva la región activa en ese momento (mediante Route 53 ARC). Si el cliente quiere conectarse a una región pasiva (por ejemplo, para operaciones de lectura), puede buscar otro nombre de DNS.

> **Note**
>
> Los clientes almacenan en caché las direcciones IP en la respuesta de Route 53 durante un tiempo indicado por la configuración de tiempo de vida (TTL) del nombre de dominio. Un TTL más largo amplía el objetivo de tiempo de recuperación (RTO) para que todos los clientes reconozcan el nuevo punto de conexión. Un valor de 60 segundos es típico para utilizar en la conmutación por error. No todo el software cumple a la perfección con el vencimiento del TTL del DNS. Además, es posible que haya varios niveles de almacenamiento en caché del DNS, por ejemplo, en el sistema operativo, la máquina virtual y la aplicación.

En lo que respecta al modo de escritura en su región, es mejor evitar Route 53 a menos que también utilice el enrutamiento de solicitudes de la capa de computación.

Enrutamiento de solicitudes de Global Accelerator

Con AWS Global Accelerator, como se muestra en el diagrama siguiente, un cliente busca el nombre de dominio conocido en Route 53. No obstante, en lugar de obtener una dirección IP que corresponda a un punto de conexión regional, el cliente recibe una dirección IP estática anycast que enruta a la ubicación periférica de AWS más cercana. A partir de esa ubicación periférica, todo el tráfico se enruta en la red privada de AWS y hacia algún punto de conexión (como un equilibrador de carga o API Gateway) en una región elegida por las reglas de enrutamiento que se mantienen en Global Accelerator. En comparación con el enrutamiento basado en las reglas de Route 53, el enrutamiento de solicitudes de Global Accelerator tiene latencias más bajas porque reduce la cantidad de tráfico en el Internet público. Además, como Global Accelerator no depende del vencimiento de TTL del DNS para cambiar las reglas de enrutamiento, puede ajustar el enrutamiento con mayor rapidez.



Con el modo de escritura en cualquier región, o si se combina con el enrutamiento de solicitudes de la capa de computación en el backend, Global Accelerator funciona a la perfección. El cliente se conecta a la ubicación periférica más cercana y no tiene por qué preocuparse de qué región recibe la solicitud.

Con la escritura en una región, las reglas de enrutamiento de Global Accelerator deben enviar las solicitudes a la región activa en ese momento. Puede utilizar comprobaciones de estado que informen artificialmente de un error en cualquier región que su sistema global no considere la activa. Al igual que con DNS, se puede utilizar un nombre de dominio DNS alternativo para enrutar las solicitudes de lectura si las solicitudes pueden proceder de cualquier región.

En lo que respecta modo de escritura en su región, es mejor evitar Global Accelerator a menos que también utilice el enrutamiento de solicitudes de la capa de computación.

Procesos de evacuación

Evacuar una región es el proceso de migrar la actividad (en general la actividad de lectura y escritura) fuera de esa región.

Evacuación de una región activa

Es posible que decida evacuar una región activa por varios motivos: como parte de la actividad empresarial habitual (por ejemplo, si utiliza un modo de seguimiento del sol, escritura en una región), debido a una decisión empresarial de cambiar la región activa en ese momento, en respuesta a

errores en la pila de software fuera de DynamoDB, o porque se encuentra con problemas generales como latencias más altas de lo habitual en la región.

Con el modo de escritura en cualquier región, evacuar una región activa es sencillo. Puede dirigir el tráfico a regiones alternativas con cualquier sistema de enrutamiento y permitir que las operaciones de escritura en la región evacuada se repliquen como de costumbre.

Los modos escribir en una región y escribir en su región se utilizan, por lo general, con las tablas de MREC. Por lo tanto, asegúrese de que todas las operaciones de escritura en la región activa se hayan registrado por completo, procesado por flujo y propagado de manera global antes de iniciar las operaciones de escritura en la nueva región activa, para garantizar que las operaciones de escritura futuras se procesen con la última versión de los datos.

Supongamos que la región A es activa y la región B es pasiva (para la tabla completa o para los elementos asignados a la región A). El mecanismo típico para llevar a cabo una evacuación consiste en pausar las operaciones de escritura en A, esperar el tiempo suficiente para que esas operaciones se hayan propagado completamente a B, actualizar la pila de la arquitectura para que reconozca B como activa y, a continuación, reanudar las operaciones de escritura en B. No existe ninguna métrica que indique con absoluta certeza que la región A ha replicado completamente sus datos a la región B. Si la región A está en buen estado, pausar las operaciones de escritura en la región A y esperar diez veces el valor máximo reciente de la métrica `ReplicationLatency` normalmente sería suficiente para determinar que la replicación se ha completado. Si el estado de la región A no es correcto y muestra otras zonas de latencias aumentadas, elegiría un múltiplo mayor para el tiempo de espera.

Evacuación de una región sin conexión

Hay que considerar un caso especial: ¿qué pasaría si la región A se desconectara por completo sin previo aviso? Esto es muy poco probable, pero se debe tener en cuenta de todos modos.

Evacuación de una tabla de MRSC desconectada

> Si esto sucede con una tabla de MRSC, no es necesario que haga nada especial. Las tablas de MRSC admiten un objetivo de punto de recuperación (RPO) de cero. Todas las operaciones de escritura correctas que se hagan en la tabla de MRSC en la región sin conexión estarán disponibles en todas las demás tablas de la región, por lo que no habrá ningún riesgo potencial de los datos incluso si la región se desconecta por completo sin previo aviso. Las empresas pueden seguir utilizando las réplicas ubicadas en las demás regiones.

Evacuación de una tabla de MREC desconectada

Si esto sucede con una tabla de MREC, las operaciones de escritura en la región A que aún no se haya propagado se retiene y se propaga después de que la región A vuelva a estar en línea. Las operaciones de escritura no se pierden, pero su propagación se retrasa indefinidamente.

La aplicación decide cómo continuar en este caso. Por motivos de continuidad empresarial, es posible que las operaciones de escritura deban continuar hacia la nueva región primaria B. No obstante, si un elemento de la región B recibe una actualización mientras hay una propagación pendiente de una operación de escritura para ese elemento desde la región A, la propagación se suprime según el modelo de último escritor gana. Cualquier actualización en la región B podría suprimir una solicitud de escritura entrante.

Con el modo de escritura en cualquier región, las operaciones de lectura y escritura pueden continuar en la región B, con la confianza de que los elementos de la región A se propagarán finalmente en la región B y la aceptación de la posibilidad de que falten elementos hasta que la región A vuelva a estar en línea. Cuando sea posible, como con las operaciones de escritura de idempotencia, debe considerar la posibilidad de reproducir el tráfico de escritura reciente (por ejemplo, con un origen de eventos ascendente) para completar las operaciones de escritura que puedan faltar y dejar que el último escritor que gane la resolución del conflicto suprima la propagación final de la operación de escritura entrante.

Con los otros modos de escritura, hay que considerar hasta qué punto se puede seguir trabajando con una visión del mundo ligeramente desactualizada. Faltará una pequeña duración de las operaciones de escritura, según el seguimiento de `ReplicationLatency`, hasta que la región A vuelva a estar en línea. ¿Puede avanzar la actividad? En algunos casos de uso puede que sí, pero en otros puede que no si no hay mecanismos de mitigación adicionales.

Por ejemplo, imagine que tiene que mantener un saldo de crédito disponible sin interrupción incluso después de una interrupción total de la región. Podría dividir el saldo en dos elementos distintos, uno ubicado en la región A y otro en la región B, e iniciar cada uno con la mitad del saldo disponible. De este modo, se utilizaría el modo de escritura en su región. Las actualizaciones transaccionales procesadas en cada región se escribirían según la copia local del saldo. Si la región A se queda totalmente fuera de línea, se podría seguir trabajando con el procesamiento de transacciones en la región B y las operaciones de escritura se limitarían a la parte del saldo que se mantiene en la región B. Dividir el saldo de esta manera conlleva complejidades cuando el saldo baja o hay que reequilibrar el crédito, pero proporciona un ejemplo de recuperación segura de la actividad incluso con operaciones de escritura pendientes inciertas.

Un ejemplo más: imagine que captura datos de un formulario web. Puede utilizar el control de simultaneidad optimista (OCC) para asignar versiones a los elementos de datos e insertar la última versión en el formulario web como un campo oculto. En cada envío, la operación de escritura solo se realiza correctamente si la versión de la base de datos sigue coincidiendo con la versión con la que se creó el formulario. Si las versiones no coinciden, el formulario web puede actualizarse (o combinarse cuidadosamente) en función de la versión actual de la base de datos y el usuario puede continuar de nuevo. El modelo OCC suele proteger contra el hecho de que otro cliente sobrescriba los datos y produzca una nueva versión de ellos, pero también puede ser de ayuda durante la conmutación por error, cuando un cliente puede encontrarse con versiones más antiguas de los datos. Imaginemos que utiliza la marca de tiempo como versión. El formulario se creó por primera vez en la región A a las 12:00, pero (tras la conmutación por error) intenta escribir en la región B y se da cuenta de que la última versión en la base de datos es la de las 11:59. En este escenario, el cliente puede esperar a que la versión de las 12:00 se propague a la región B y escribir en esa versión, o basarse en la de las 11:59 y crear una nueva versión de las 12:01 (que, después de la escritura, suprimiría la versión entrante una vez que se recupere la región A).

Un tercer ejemplo: una empresa de servicios financieros almacena los datos sobre las cuentas de sus clientes y sus transacciones financieras en una base de datos de DynamoDB. En caso de que se produzca una interrupción completa en la región A, quiere asegurarse de que las actividades de escritura relacionadas con sus cuentas estén totalmente disponibles en la región B o bien quiere poner en cuarentena sus cuentas como parciales conocidas hasta que la región A vuelva a estar en línea. En lugar de pausar toda la actividad, decide pausarla solo para la minúscula fracción de cuentas que ha determinado que tienen transacciones no propagadas. Para lograrlo, utilizan una tercera región, a la que llamaremos región C. Antes de procesar cualquier operación de escritura en la región A, incluyen un resumen sucinto de esas operaciones pendientes (por ejemplo, un nuevo recuento de transacciones para una cuenta) en la región C. Este resumen es suficiente para que la región B determine si su vista está totalmente actualizada. Esta acción bloquea de un modo efectivo la cuenta desde el momento de la escritura en la región C hasta que la región A acepta las operaciones de escritura y la región B las recibe. Los datos de la región C no se utilizan excepto como parte de un proceso de conmutación por error, tras el cual la región B puede cruzar sus datos con los de la región C para comprobar si alguna de sus cuentas está desactualizada. Esas cuentas se marcarían como en cuarentena hasta que la recuperación de la región A propague los datos parciales a la región B. Si se produce un error en la región C, se puede crear una nueva región D para utilizarla en su lugar. Los datos de la región C son muy temporales y, al cabo de unos minutos, la región D dispondrá de un registro lo suficientemente actualizado de las operaciones de escritura en tránsito como para ser totalmente

útil. Si se produce un error en la región B, la región A puede seguir aceptando solicitudes de escritura en cooperación con la región C. Esta empresa estaba dispuesta a aceptar escrituras de latencia más alta (a dos regiones: C y luego A) y tuvo la suerte de contar con un modelo de datos en el que se podía resumir sucintamente el estado de una cuenta.

Planificación de la capacidad de rendimiento para tablas globales de DynamoDB

La migración del tráfico de una región a otra requiere un examen cuidadoso de la configuración de las tablas de DynamoDB en lo que respecta a la capacidad.

Estas son algunas consideraciones de la administración de la capacidad de escritura:

- Una tabla global debe estar en modo bajo demanda o aprovisionada con el escalado automático activado.
- Si se aprovisiona con escalado automático, la configuración de escritura (utilización mínima, máxima y objetivo) se replica en todas las regiones. Aunque la configuración del escalado automático esté sincronizada, la capacidad de escritura real aprovisionada puede flotar independientemente entre las regiones.
- Una de las razones por las que puede ver una capacidad de escritura aprovisionada diferente se debe a la característica TTL. Al activar TTL en DynamoDB, puede especificar un nombre de atributo cuyo valor indique el tiempo de caducidad del elemento, en formato de tiempo de época Unix en segundos. Transcurrido ese tiempo, DynamoDB puede eliminar el elemento sin incurrir en costos de escritura. Con las tablas globales, puede configurar TTL en cualquier región y la configuración se replica automáticamente a otras regiones que están asociadas a la tabla global. Cuando un elemento reúne las condiciones para eliminarse mediante una regla TTL, esa tarea puede realizarse en cualquier región. La operación de eliminación se realiza sin consumir unidades de escritura en la tabla de origen, pero las tablas de réplica obtendrán una escritura replicada de esa operación de eliminación e incurrirán en costos de unidades de escritura replicadas. Las tablas de MRSC no admiten TTL.
- Si utiliza el escalado automático, asegúrese de que la configuración de la capacidad máxima de escritura aprovisionada es lo suficientemente alta como para gestionar todas las operaciones de escritura, así como todas las posibles operaciones de eliminación de TTL. El escalado automático ajusta cada región en función de su consumo de escritura. Las tablas bajo demanda no tienen una configuración de capacidad de escritura máxima aprovisionada, pero el límite de rendimiento de escritura máxima en el nivel de tabla especifica la capacidad de escritura máxima sostenida

que permitirá la tabla bajo demanda. El límite predeterminado es de 40 000, pero se puede ajustar. Le recomendamos que lo establezca lo suficientemente alto como para gestionar todas las operaciones de escritura (incluidas las operaciones de escritura TTL) que pueda necesitar la tabla bajo demanda. Este valor debe ser el mismo en todas las regiones participantes cuando configure tablas globales.

Estas son algunas consideraciones de la administración de la capacidad de lectura:

- Se permite que la configuración de la administración de la capacidad de lectura difiera de una región a otra porque se supone que las distintas regiones pueden tener patrones de lectura independientes. Al agregar por primera vez una réplica global a una tabla, se propaga la capacidad de la región de origen. Tras la creación, puede ajustar la configuración de la capacidad de lectura, que no se transfiere al otro lado.
- Cuando utilice el escalado automático de DynamoDB, asegúrese de que la configuración de la capacidad máxima de lectura aprovisionada es lo suficientemente alta como para gestionar todas las operaciones de lectura en todas las regiones. Durante las operaciones estándar, la capacidad de lectura quizá se reparta entre las regiones, pero durante la conmutación por error la tabla debería poder adaptarse automáticamente al aumento de la carga de trabajo de lectura. Las tablas bajo demanda no tienen una configuración de capacidad de lectura máxima aprovisionada, pero el límite de rendimiento de lectura máxima en el nivel de tabla especifica la capacidad de lectura máxima sostenida que permitirá la tabla bajo demanda. El límite predeterminado es de 40 000, pero se puede ajustar. Le recomendamos que lo establezca lo suficientemente alto como para gestionar todas las operaciones de lectura que podría necesitar la tabla si todas las operaciones de lectura tuvieran que enrutarse a esta única región.
- Si una tabla de una región no suele recibir tráfico de lectura, pero podría tener que absorber una gran cantidad de tráfico de lectura tras una conmutación por error, puede preparar la capacidad para que acepte un nivel de tráfico de lectura más alto.

ARC tiene comprobaciones de preparación que pueden ser útiles para confirmar que las regiones de DynamoDB tienen configuraciones de tabla y cuotas de cuenta similares, si utiliza Route 53 para enrutar solicitudes o si no lo utiliza. Estas comprobaciones de preparación también pueden ser de ayuda a la hora de ajustar las cuotas en el nivel de cuenta para asegurarse de que coinciden.

Lista de comprobación de preparación para tablas globales de DynamoDB

Utilice la siguiente lista de comprobación para tomar decisiones y realizar tareas cuando despliegue tablas globales.

- Determine si su caso de uso se beneficia más de un modo de coherencia MRSC o MREC. ¿Necesita una coherencia sólida, incluso con una latencia más alta y otras desventajas?
- Determine cuántas y qué regiones deben participar en la tabla global. Si planea utilizar MRSC, decida si quiere que la tercera región sea una réplica o un testigo.
- Determine el modo de escritura de la aplicación. No es lo mismo que el modo de coherencia. Para obtener más información, consulte Modos de escritura con tablas globales de DynamoDB.
- Planifique su estrategia de Estrategias de enrutamiento de DynamoDB en función de su modo de escritura.
- Defina

Evacuar una región es el proceso de migrar la actividad (en general la actividad de lectura y escritura) fuera de esa región.

Evacuación de una región activa

Es posible que decida evacuar una región activa por varios motivos: como parte de la actividad empresarial habitual (por ejemplo, si utiliza un modo de seguimiento del sol, escritura en una región), debido a una decisión empresarial de cambiar la región activa en ese momento, en respuesta a errores en la pila de software fuera de DynamoDB, o porque se encuentra con problemas generales como latencias más altas de lo habitual en la región.

Con el modo de escritura en cualquier región, evacuar una región activa es sencillo. Puede dirigir el tráfico a regiones alternativas con cualquier sistema de enrutamiento y permitir que las operaciones de escritura en la región evacuada se repliquen como de costumbre.

Los modos escribir en una región y escribir en su región se utilizan, por lo general, con las tablas de MREC. Por lo tanto, asegúrese de que todas las operaciones de escritura en la región activa se hayan registrado por completo, procesado por flujo y propagado de manera global antes de iniciar las operaciones de escritura en la nueva región activa, para garantizar que las operaciones de escritura futuras se procesen con la última versión de los datos.

Supongamos que la región A es activa y la región B es pasiva (para la tabla completa o para los elementos asignados a la región A). El mecanismo típico para llevar a cabo una

evacuación consiste en pausar las operaciones de escritura en A, esperar el tiempo suficiente para que esas operaciones se hayan propagado completamente a B, actualizar la pila de la arquitectura para que reconozca B como activa y, a continuación, reanudar las operaciones de escritura en B. No existe ninguna métrica que indique con absoluta certeza que la región A ha replicado completamente sus datos a la región B. Si la región A está en buen estado, pausar las operaciones de escritura en la región A y esperar diez veces el valor máximo reciente de la métrica `ReplicationLatency` normalmente sería suficiente para determinar que la replicación se ha completado. Si el estado de la región A no es correcto y muestra otras zonas de latencias aumentadas, elegiría un múltiplo mayor para el tiempo de espera.

Evacuación de una región sin conexión

Hay que considerar un caso especial: ¿qué pasaría si la región A se desconectara por completo sin previo aviso? Esto es muy poco probable, pero se debe tener en cuenta de todos modos.

Evacuación de una tabla de MRSC desconectada

Si esto sucede con una tabla de MRSC, no es necesario que haga nada especial. Las tablas de MRSC admiten un objetivo de punto de recuperación (RPO) de cero. Todas las operaciones de escritura correctas que se hagan en la tabla de MRSC en la región sin conexión estarán disponibles en todas las demás tablas de la región, por lo que no habrá ningún riesgo potencial de los datos incluso si la región se desconecta por completo sin previo aviso. Las empresas pueden seguir utilizando las réplicas ubicadas en las demás regiones.

Evacuación de una tabla de MREC desconectada

Si esto sucede con una tabla de MREC, las operaciones de escritura en la región A que aún no se haya propagado se retiene y se propaga después de que la región A vuelva a estar en línea. Las operaciones de escritura no se pierden, pero su propagación se retrasa indefinidamente.

La aplicación decide cómo continuar en este caso. Por motivos de continuidad empresarial, es posible que las operaciones de escritura deban continuar hacia la nueva región primaria B. No obstante, si un elemento de la región B recibe una actualización mientras hay una propagación pendiente de una operación de escritura para ese elemento desde la región A, la propagación se suprime según el modelo de último escritor gana. Cualquier actualización en la región B podría suprimir una solicitud de escritura entrante.

Con el modo de escritura en cualquier región, las operaciones de lectura y escritura pueden continuar en la región B, con la confianza de que los elementos de la región A se propagarán finalmente en la región B y la aceptación de la posibilidad de que falten elementos hasta que la región A vuelva a estar en línea. Cuando sea posible, como con las operaciones de escritura de idempotencia, debe considerar la posibilidad de reproducir el tráfico de escritura reciente (por ejemplo, con un origen de eventos ascendente) para completar las operaciones de escritura que puedan faltar y dejar que el último escritor que gane la resolución del conflicto suprima la propagación final de la operación de escritura entrante.

Con los otros modos de escritura, hay que considerar hasta qué punto se puede seguir trabajando con una visión del mundo ligeramente desactualizada. Faltará una pequeña duración de las operaciones de escritura, según el seguimiento de `ReplicationLatency`, hasta que la región A vuelva a estar en línea. ¿Puede avanzar la actividad? En algunos casos de uso puede que sí, pero en otros puede que no si no hay mecanismos de mitigación adicionales.

Por ejemplo, imagine que tiene que mantener un saldo de crédito disponible sin interrupción incluso después de una interrupción total de la región. Podría dividir el saldo en dos elementos distintos, uno ubicado en la región A y otro en la región B, e iniciar cada uno con la mitad del saldo disponible. De este modo, se utilizaría el modo de escritura en su región. Las actualizaciones transaccionales procesadas en cada región se escribirían según la copia local del saldo. Si la región A se queda totalmente fuera de línea, se podría seguir trabajando con el procesamiento de transacciones en la región B y las operaciones de escritura se limitarían a la parte del saldo que se mantiene en la región B. Dividir el saldo de esta manera conlleva complejidades cuando el saldo baja o hay que reequilibrar el crédito, pero proporciona un ejemplo de recuperación segura de la actividad incluso con operaciones de escritura pendientes inciertas.

Un ejemplo más: imagine que captura datos de un formulario web. Puede utilizar el control de simultaneidad optimista (OCC) para asignar versiones a los elementos de datos e insertar la última versión en el formulario web como un campo oculto. En cada envío, la operación de escritura solo se realiza correctamente si la versión de la base de datos sigue coincidiendo con la versión con la que se creó el formulario. Si las versiones no coinciden, el formulario web puede actualizarse (o combinarse cuidadosamente) en función de la versión actual de la base de datos y el usuario puede continuar de nuevo. El modelo OCC suele proteger contra el hecho de que otro cliente sobrescriba los datos y produzca una nueva versión de ellos, pero también puede ser de ayuda durante la conmutación por error, cuando un cliente puede

encontrarse con versiones más antiguas de los datos. Imaginemos que utiliza la marca de tiempo como versión. El formulario se creó por primera vez en la región A a las 12:00, pero (tras la conmutación por error) intenta escribir en la región B y se da cuenta de que la última versión en la base de datos es la de las 11:59. En este escenario, el cliente puede esperar a que la versión de las 12:00 se propague a la región B y escribir en esa versión, o basarse en la de las 11:59 y crear una nueva versión de las 12:01 (que, después de la escritura, suprimiría la versión entrante una vez que se recupere la región A).

Un tercer ejemplo: una empresa de servicios financieros almacena los datos sobre las cuentas de sus clientes y sus transacciones financieras en una base de datos de DynamoDB. En caso de que se produzca una interrupción completa en la región A, quiere asegurarse de que las actividades de escritura relacionadas con sus cuentas estén totalmente disponibles en la región B o bien quiere poner en cuarentena sus cuentas como parciales conocidas hasta que la región A vuelva a estar en línea. En lugar de pausar toda la actividad, decide pausarla solo para la minúscula fracción de cuentas que ha determinado que tienen transacciones no propagadas. Para lograrlo, utilizan una tercera región, a la que llamaremos región C. Antes de procesar cualquier operación de escritura en la región A, incluyen un resumen sucinto de esas operaciones pendientes (por ejemplo, un nuevo recuento de transacciones para una cuenta) en la región C. Este resumen es suficiente para que la región B determine si su vista está totalmente actualizada. Esta acción bloquea de un modo efectivo la cuenta desde el momento de la escritura en la región C hasta que la región A acepta las operaciones de escritura y la región B las recibe. Los datos de la región C no se utilizan excepto como parte de un proceso de conmutación por error, tras el cual la región B puede cruzar sus datos con los de la región C para comprobar si alguna de sus cuentas está desactualizada. Esas cuentas se marcarían como en cuarentena hasta que la recuperación de la región A propague los datos parciales a la región B. Si se produce un error en la región C, se puede crear una nueva región D para utilizarla en su lugar. Los datos de la región C son muy temporales y, al cabo de unos minutos, la región D dispondrá de un registro lo suficientemente actualizado de las operaciones de escritura en tránsito como para ser totalmente útil. Si se produce un error en la región B, la región A puede seguir aceptando solicitudes de escritura en cooperación con la región C. Esta empresa estaba dispuesta a aceptar escrituras de latencia más alta (a dos regiones: C y luego A) y tuvo la suerte de contar con un modelo de datos en el que se podía resumir sucintamente el estado de una cuenta.

the section called “Procesos de evacuación”, según el modo de coherencia, el modo de escritura y la estrategia de enrutamiento.

- Capture métricas sobre el estado, la latencia y los errores de cada región. Para obtener una lista de las métricas de DynamoDB, consulte la entrada de blog de AWS Supervisión de Amazon DynamoDB para el reconocimiento operativo, donde encontrará una lista de las métricas que debe examinar. También debe utilizar valores controlados sintéticos (solicitudes artificiales diseñadas para detectar errores; en inglés se llama “canary”, por el uso que se hacía de los canarios en las minas de carbón), así como la observación en directo del tráfico de los clientes. En las métricas de DynamoDB no aparecerán todas las incidencias.
- Si utiliza MREC, establezca alarmas para los aumentos sostenidos de `ReplicationLatency`. Un aumento podría indicar un error de configuración accidental en el que la tabla global tiene diferentes opciones de escritura en distintas regiones, lo que da lugar a solicitudes replicadas con errores y a un aumento de las latencias. También podría indicar que existe una interrupción regional. Un buen ejemplo sería generar una alerta si el promedio reciente supera los 180 000 milisegundos. También puede vigilar si `ReplicationLatency` cae a 0, lo que indica que la replicación se ha estancado.
- Asigne una configuración máxima de lectura y escritura suficiente para cada tabla global.
- Identifique con antelación las razones para evacuar una región. Si la decisión implica una evaluación manual, documente todas las consideraciones. Este trabajo debe realizarse cuidadosamente con antelación, no bajo estrés.
- Mantenga un manual de procedimientos para cada acción que deba llevarse a cabo cuando evacúe una región. Normalmente se requiere muy poco trabajo para las tablas globales, pero trasladar el resto de la pila puede resultar complejo.

> **Note**
>
> Con los procedimientos de conmutación por error, es una práctica recomendada confiar solo en las operaciones del plano de datos y no en las del plano de control, porque algunas operaciones del plano de control pueden deteriorarse durante los errores de la región.

Para obtener más información, consulte la entrada del blog de AWS Crear aplicaciones resilientes con tablas globales de Amazon DynamoDB: parte 4.

- Pruebe periódicamente todos los aspectos del manual de procedimientos, incluidas las evacuaciones de región. Un manual de procedimientos no probado es un manual poco fiable.
- Considere la posibilidad de utilizar AWS Resilience Hub para evaluar la resiliencia de toda la aplicación (incluidas las tablas globales). Proporciona una visión completa del estado de resiliencia general de su cartera de aplicaciones a través de su panel.
- Considere la posibilidad de utilizar las comprobaciones de preparación de ARC para evaluar la configuración actual de la aplicación y realizar un seguimiento de cualquier desviación de las prácticas recomendadas.
- Al escribir comprobaciones de estado para utilizarlas con Route 53 o Global Accelerator, haga una serie de llamadas que cubran todo el flujo de base de datos. Si limita la comprobación para confirmar únicamente que el punto de conexión de DynamoDB está activo, no se podrán cubrir muchos modos de error, como los errores de configuración de AWS Identity and Access Management (IAM), los problemas de implementación del código, los errores en la pila fuera de DynamoDB, las latencias de lectura o escritura superiores al promedio, etc.

Preguntas frecuentes sobre el despliegue de tablas globales

Cuál es el precio de las tablas globales?

- El precio de una operación de escritura en una tabla de DynamoDB tradicional se calcula en unidades de capacidad de escritura (WCU, para tablas aprovisionadas) o unidades de solicitud de escritura (WRU) para tablas bajo demanda. Si escribe un elemento de 5 KB, se genera un cargo de 5 unidades. Una operación de escritura en una tabla global se cobra en unidades de capacidad de escritura replicada (rWCU, para las tablas aprovisionadas) o en unidades de solicitud de escritura replicada (rWRU, para las tablas bajo demanda). Las rWCU y las rWRU tienen el mismo precio que las WGU y las WRU.
- Los cargos por rWCU y rWRU se producen en cada región en la que se escribe el elemento de manera directa o mediante replicación. Se aplican tarifas de transferencia de datos entre regiones.
- La escritura en un índice secundario global (GSI) se considera una operación de escritura local y utiliza unidades de escritura normales.
- No hay capacidad reservada disponible para las rWCU ni rWRU de momento. Adquirir capacidad reservada para WCU puede ser beneficioso para las tablas con GSI que consumen unidades de escritura.

- Al agregar una nueva región a una tabla global, DynamoDB inicia la nueva región automáticamente y le cobra como si se tratara de una restauración de tabla, en función del tamaño de GB de la tabla. También cobra tarifas de transferencia de datos entre regiones.

Qué regiones admiten las tablas globales?

Las tablas globales versión 2019.11.21 (actual) admiten todas las Regiones de AWS para las tablas de MREC y los siguientes conjuntos de regiones para las tablas de MRSC:

- Conjunto de regiones de EE. UU.: Este de EE. UU. (Norte de Virginia), Este de EE. UU. (Ohio), Oeste de EE. UU. (Oregón)
- Conjunto de regiones de la Unión Europea: Europa (Irlanda), Europa (Londres), Europa (París), Europa (Fráncfort)
- Conjunto de regiones de Asia-Pacífico: Asia-Pacífico (Tokio), Asia-Pacífico (Seúl) y Asia-Pacífico (Osaka).

Cómo se gestionan los GSI con las tablas globales?

En la versión 2019.11.21 (actual) de las tablas globales, cuando se crea un GSI en una región, se crea automáticamente en otras regiones participantes y se repone de forma automática.

Cómo detengo la replicación de una tabla global?

- Puede eliminar una tabla de réplica del mismo modo que eliminaría cualquier otra tabla. Al eliminar la tabla global se detiene la replicación en esa región y se elimina la copia de la tabla guardada en dicha región. No obstante, no se puede detener la replicación mientras se mantienen copias de la tabla como entidades independientes, ni tampoco se puede pausar.
- Una tabla de MRSC se debe implementar exactamente en tres regiones. Para eliminar las réplicas, debe eliminar todas las réplicas y el testigo para que la tabla de MRSC pase a ser una tabla local.

Cómo interactúan los flujos de DynamoDB con las tablas globales?

- Cada tabla global produce un flujo independiente basado en todas sus operaciones de escritura, independientemente de dónde hayan comenzado. Puede elegir consumir el flujo de DynamoDB en una región o en todas las regiones (de forma independiente). Si desea procesar operaciones de escritura locales pero no replicadas, puede agregar su propio atributo de región a cada elemento para identificar la región de escritura. A continuación, puede utilizar un filtro de eventos Lambda

para llamar la función de Lambda solo para las operaciones de escritura en la región local. Esta acción ayuda en las operaciones de inserción y actualización, pero no en las de eliminación.

- Las tablas globales configuradas para una coherencia final de varias regiones (MREC) replican los cambios al leer esos cambios desde un flujo de DynamoDB en una tabla de réplica y aplicarlos a todas las demás tablas de réplica. Por lo tanto, DynamoDB está habilitado de forma predeterminada en todas las réplicas de una tabla global de MREC y no se puede desactivar en esas réplicas. El proceso de replicación de MREC puede combinar varios cambios en un periodo breve en una sola operación de escritura replicada. Como resultado, el flujo de cada réplica puede contener registros ligeramente distintos. Los registros de DynamoDB Streams en las réplicas de MREC siempre se ordenan por elemento, pero el orden entre los elementos podría variar entre las réplicas.
- Las tablas globales configuradas para una coherencia alta de varias regiones (tablas de MRSC) no utilizan DynamoDB Streams para la replicación, por lo que esta característica no está habilitada de manera predeterminada en las réplicas de MRSC. Puede habilitar DynamoDB Streams en una réplica de MRSC. Los registros de DynamoDB Streams en las réplicas de MRSC son idénticos para todas las réplicas y siempre se ordenan por elemento, pero el orden entre los elementos podría variar entre las réplicas.

Cómo gestionan las transacciones las tablas globales?

- Las operaciones transaccionales en las tablas de la MRSC generarán errores.
- Las operaciones transaccionales en las tablas de MREC proporcionan garantías de atomicidad, coherencia, aislamiento y durabilidad (ACID) solo en la región en la que se creó la operación de escritura originalmente. No se admiten las transacciones entre regiones en las tablas globales. Por ejemplo, si tiene una tabla global de MREC con réplicas en las regiones Este de EE. UU. (Ohio) y Oeste de EE. UU. (Oregón) y hace una operación `TransactWriteItems` en la región Este de EE. UU. (Ohio), puede observar transacciones completadas de manera parcial en la región Oeste de EE. UU. (Oregón) a medida que los cambios se replican. Los cambios se replican en otras regiones solo cuando se han confirmado en la región de origen.

Cómo interactúan las tablas globales con la memoria caché de DynamoDB Accelerator (DAX)?

Las tablas globales eluden DAX mediante la actualización directa de DynamoDB, por lo que DAX no tiene constancia de que está almacenando datos obsoletos. La memoria caché de DAX solo se actualiza cuando caduca el TTL de la memoria caché.

Se propagan las etiquetas de las tablas?

No, las etiquetas no se propagan automáticamente.

Debo hacer copias de seguridad de las tablas de todas las regiones o solo de una?

La respuesta depende de la finalidad de la copia de seguridad.

- Si desea garantizar la durabilidad de los datos, DynamoDB ya proporciona esa protección. El servicio garantiza la durabilidad.
- Si desea conservar una instantánea para los registros históricos (por ejemplo, para cumplir los requisitos normativos), la copia de seguridad en una región debería ser suficiente. Puede copiar la copia de seguridad a más regiones mediante AWS Backup.
- Si desea recuperar datos eliminados o modificados erróneamente, utilice la recuperación en un momento dado (PITR) de DynamoDB en una región.

Cómo puedo desplegar tablas globales con CloudFormation?

- CloudFormation representa una tabla de DynamoDB y una tabla global como dos recursos independientes: `AWS::DynamoDB::Table` y `AWS::DynamoDB::GlobalTable`. Un enfoque consiste en crear todas las tablas que puedan ser potencialmente globales mediante el constructo `GlobalTable`, mantenerlas inicialmente como tablas independientes y agregar regiones después, si es necesario.
- En CloudFormation, cada tabla global está controlada por una sola pila, en una sola región, independientemente del número de réplicas. Cuando implemente la plantilla, CloudFormation crea y actualiza todas las réplicas como parte de una sola operación de pila. No debe desplegar el mismo recurso AWS::DynamoDB::GlobalTable en varias regiones. Se producirán errores y no se admite. Si despliega la plantilla de aplicación en varias regiones, puede usar condiciones para crear el recurso `AWS::DynamoDB::GlobalTable` en una sola región. O bien, puede optar por definir recursos `AWS::DynamoDB::GlobalTable` en una pila que sea independiente de la pila de aplicaciones y asegurarse de que despliega en una sola región.
- Si tiene una tabla normal y desea convertirla en una tabla global sin que CloudFormation deje de administrarla, establezca la política de eliminación en `Retain`, elimine la tabla de la pila, conviértala en una tabla global en la consola y, a continuación, importe la tabla global como un nuevo recurso a la pila. Para más información, consulte el repositorio de GitHub de AWS.
- En este momento no se admite la replicación entre cuentas.

Conclusión y recursos

Las tablas globales de DynamoDB tienen muy pocos controles, aunque todavía es necesario prestarles atención especial. Debe determinar su modo de escritura, el modelo de enrutamiento y los procesos de evacuación. Debe instrumentar la aplicación en todas las regiones y estar preparado para ajustar el enrutamiento o realizar una evacuación para mantener el estado global. La recompensa es disponer de un conjunto de datos distribuidos de manera global con operaciones de lectura y escritura de baja latencia y diseñado para una disponibilidad del 99,999 %.

Para más información acerca de las tablas globales de DynamoDB, consulte los recursos siguientes:

- Documentación sobre DynamoDB
- Controlador de recuperación de aplicaciones de Amazon
- Readiness check in ARC (documentación de AWS)
- Políticas de enrutamiento de Route 53
- AWS Global Accelerator
- Acuerdo de nivel de servicio para Amazon DynamoDB
- AWS Multi-Region Fundamentals (documento técnico de AWS)
- Data resiliency design patterns with AWS (presentación de AWS re:Invent 2022)
- How Fidelity Investments and Reltio modernized with Amazon DynamoDB (presentación de AWS re:Invent 2022)
- Multi-Region design patterns and best practices (presentación de AWS re:Invent 2022)
- Disaster Recovery (DR) Architecture on AWS, Part III: Pilot Light and Warm Standby (entrada del blog de AWS)
- Use Region pinning to set a home Region for items in an Amazon DynamoDB global table (entrada del blog de AWS)
- Monitoring Amazon DynamoDB for operational awareness (entrada del blog de AWS)
- Scaling DynamoDB: How partitions, hot keys, and split for heat impact performance (entrada del blog de AWS)
- Multi-Region strong consistency with DynamoDB global tables (presentación de AWS re:Invent 2024)

Prácticas recomendadas para administrar el plano de control en DynamoDB

> **Note**
>
> DynamoDB ingresa una limitación del plano de control de 2500 solicitudes por segundo con la opción de volver a intentarlo. Consulte a continuación para obtener detalles adicionales.

Las operaciones del plano de control de DynamoDB permiten administrar las tablas de DynamoDB y los objetos que dependen de las tablas, como los índices. Para obtener más información sobre estas operaciones, consulte Plano de control.

En algunas circunstancias, es posible que deba tomar medidas y utilizar los datos devueltos por las llamadas de plano de control como parte de su lógica empresarial. Por ejemplo, es posible que necesite saber el valor de `ProvisionedThroughput` devuelto por `DescribeTable`. En estas circunstancias, siga estas prácticas recomendadas:

- No consulte excesivamente el plano de control de DynamoDB.
- No mezcle las llamadas de plano de control y las llamadas de plano de datos dentro del mismo código.
- Controle las limitaciones en las solicitudes del plano de control y vuelva a intentarlo con un retardo.
- Invoque y realice un seguimiento de los cambios en un recurso concreto desde un único cliente.
- En lugar de recuperar los datos de la misma tabla varias veces a intervalos cortos, almacene en caché los datos para procesarlos.

Prácticas recomendadas para utilizar operaciones de datos masivos en DynamoDB

DynamoDB admite operaciones por lotes, como utilizar `BatchWriteItem`, que puede realizar hasta 25 solicitudes `PutItem` y `DeleteItem` juntas. Sin embargo, `BatchWriteItem` no admite operaciones `UpdateItem`. Cuando se trata de actualizaciones masivas, la distinción radica en los requisitos y la naturaleza de la actualización. Puede utilizar otras API de DynamoDB, como `TransactWriteItems` para tamaños de lote de hasta 100. Cuando se trata de más elementos, puede utilizar servicios como AWS Glue, Amazon EMR o AWS Step Functions, o utilizar scripts y

herramientas personalizados como el intérprete de comandos de DynamoDB para actualizaciones masivas.

Temas

- Actualización por lotes condicional
- Operaciones masivas eficientes

Actualización por lotes condicional

DynamoDB admite operaciones por lotes, como `BatchWriteItem`, que puede realizar hasta 25 solicitudes `PutItem` y `DeleteItem` en un solo lote. No obstante, `BatchWriteItem` no admite operaciones `UpdateItem` ni expresiones condicionales. Como solución alternativa, puede utilizar otras API de DynamoDB, como `TransactWriteItems` para tamaños de lote de hasta 100.

Cuando se trata de más elementos y es necesario cambiar una gran cantidad de datos, puede utilizar servicios como AWS Glue, Amazon EMR o AWS Step Functions, o utilizar scripts y herramientas personalizados como el intérprete de comandos de DynamoDB para realizar actualizaciones masivas de forma eficiente.

Cuándo utilizar este patrón

- El intérprete de comandos de DynamoDB no es compatible con el caso de uso de producción.
- `TransactWriteItems`: hasta 100 actualizaciones individuales con condiciones o sin ellas, ejecutándose como una agrupación ACID de todo o nada. Las llamadas a `TransactWriteItems` también pueden suministrarse con un `ClientRequestToken` si la aplicación requiere idempotencia, lo que significa que varias llamadas idénticas tienen el mismo efecto que una sola llamada. Esto garantiza que no ejecute la misma transacción varias veces y acabe con un estado de datos incorrecto.

  Desventaja: se consume rendimiento adicional. 2 WCU por 1 KB de escritura en lugar del estándar de 1 WGU por 1 KB de escritura.
- PartiQL `BatchExecuteStatement`: hasta 25 actualizaciones con condiciones o sin ellas. `BatchExecuteStatement` siempre devuelve una respuesta correcta a la solicitud general y también devuelve una lista de respuestas de operaciones individuales que conserva el orden.

  Desventaja: para lotes más grandes, se requiere lógica adicional del cliente para distribuir las solicitudes en lotes de 25. Deben tenerse en cuenta las respuestas individuales a los errores para determinar la estrategia de reintentos.

Ejemplos de código

En estos ejemplos de código se utiliza la biblioteca boto3, que es el AWS SDK para Python. En los ejemplos se supone que tiene boto3 instalado y configurado con las credenciales de AWS adecuadas.

Imagine una base de datos de inventario para un vendedor de electrodomésticos que tiene varios almacenes en diferentes ciudades europeas. Debido a que es el final del verano, el vendedor desea retirar los ventiladores de mesa a fin de dejar espacio para otras existencias. El vendedor quiere ofrecer un descuento en el precio de todos los ventiladores de mesa suministrados desde los almacenes de Italia, pero solo si tienen un stock de reserva de 20 ventiladores de mesa. La tabla de DynamoDB se llama inventory, tiene un esquema de clave de partición sku que es un identificador único para cada producto y una clave de clasificación warehouse que es un identificador de un almacén.

En el siguiente código Python se muestra cómo realizar esta actualización por lotes condicional a través de una llamada a la API `BatchExecuteStatement`.

```
import boto3

client=boto3.client("dynamodb")

before_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("Before update: ", before_image['Items'])

response=client.batch_execute_statement(
        Statements=[
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITTUR1'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITROM1'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITROM2'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITROM5'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
```

```
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITVEN1'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITVEN2'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
            {'Statement': 'UPDATE inventory SET price=price-5 WHERE sku=? AND
 warehouse=? AND quantity > 20', 'Parameters': [{'S':'F123'}, {'S':'WITVEN3'}],
 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'},
        ],
        ReturnConsumedCapacity='TOTAL'
    )

after_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("After update: ", after_image['Items'])
```

La ejecución produce el siguiente resultado en los datos de muestra:

```
Before update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price':
 {'N': '40'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S':
 'WITROM2'}, 'price': {'N': '40'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '28'}, 'warehouse': {'S': 'WITROM5'}, 'price': {'N': '38'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '40'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price':
 {'N': '38'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S':
 'WITVEN2'}, 'price': {'N': '38'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'},
 'warehouse': {'S': 'WITVEN3'}, 'price': {'N': '35'}, 'sku': {'S': 'F123'}}]
After update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price': {'N':
 '40'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S': 'WITROM2'},
 'price': {'N': '35'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '28'}, 'warehouse':
 {'S': 'WITROM5'}, 'price': {'N': '33'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '35'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price': {'N': '38'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S': 'WITVEN2'}, 'price':
 {'N': '38'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'}, 'warehouse': {'S':
 'WITVEN3'}, 'price': {'N': '30'}, 'sku': {'S': 'F123'}}]
```

Como se trata de una operación delimitada para un sistema interno, no se han tenido en cuenta los requisitos de idempotencia. Es posible colocar barreras de protección adicionales, como que la actualización de precios solo se produzca si el precio es superior a 35 y menor de 40, para que las actualizaciones sean más sólidas.

Como alternativa, podemos realizar la misma operación de actualización por lotes mediante `TransactWriteItems` en caso de que haya requisitos de idempotencia y ACID más estrictos. No obstante, es importante recordar que, o bien se realizan todas las operaciones de la agrupación de transacciones, o bien falla todo la agrupación.

Supongamos un caso en el que hay una ola de calor en Italia y la demanda de ventiladores de mesa ha aumentado considerablemente. El proveedor quiere aumentar en 20 EUR el costo de los ventiladores de mesa que salen de cada almacén en Italia, pero el organismo regulador solo permite este aumento de costos si el costo actual es inferior a 70 EUR en todo su inventario. Es esencial que el precio se actualice en todo el inventario de una sola vez y solo una vez y únicamente si el costo es inferior a 70 EUR en cada uno de sus almacenes.

En el siguiente código Python se muestra cómo realizar esta actualización por lotes a través de una llamada a la API `TransactWriteItems`.

```
import boto3

client=boto3.client("dynamodb")

before_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("Before update: ", before_image['Items'])

response=client.transact_write_items(
        ClientRequestToken='UUIDAWS124',
        TransactItems=[
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITTUR1'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITROM1'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
```

```
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITROM2'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITROM5'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITVEN1'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITVEN2'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
            {'Update': { 'Key': {'sku': {'S':'F123'}, 'warehouse': {'S':'WITVEN3'}},
 'UpdateExpression': 'SET price = price + :inc', 'ConditionExpression': 'price < :cap',
 'ExpressionAttributeValues': { ':inc': {'N': '20'}, ':cap': {'N': '70'}}, 'TableName':
 'inventory', 'ReturnValuesOnConditionCheckFailure': 'ALL_OLD'}},
        ],
        ReturnConsumedCapacity='TOTAL'
    )

after_image=client.query(TableName='inventory', KeyConditionExpression='sku=:pk_val
 AND begins_with(warehouse, :sk_val)', ExpressionAttributeValues={':pk_val':
{'S':'F123'},':sk_val':{'S':'WIT'}},
 ProjectionExpression='sku,warehouse,quantity,price')
print("After update: ", after_image['Items'])
```

La ejecución produce el siguiente resultado en los datos de muestra:

```
Before update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price':
 {'N': '60'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S':
 'WITROM2'}, 'price': {'N': '55'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '28'}, 'warehouse': {'S': 'WITROM5'}, 'price': {'N': '53'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '55'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price':
 {'N': '58'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S':
 'WITVEN2'}, 'price': {'N': '58'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'},
 'warehouse': {'S': 'WITVEN3'}, 'price': {'N': '50'}, 'sku': {'S': 'F123'}}]
```

```
After update:  [{'quantity': {'N': '20'}, 'warehouse': {'S': 'WITROM1'}, 'price': {'N':
 '80'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '25'}, 'warehouse': {'S': 'WITROM2'},
 'price': {'N': '75'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '28'}, 'warehouse':
 {'S': 'WITROM5'}, 'price': {'N': '73'}, 'sku': {'S': 'F123'}}, {'quantity': {'N':
 '26'}, 'warehouse': {'S': 'WITTUR1'}, 'price': {'N': '75'}, 'sku': {'S': 'F123'}},
 {'quantity': {'N': '10'}, 'warehouse': {'S': 'WITVEN1'}, 'price': {'N': '78'}, 'sku':
 {'S': 'F123'}}, {'quantity': {'N': '20'}, 'warehouse': {'S': 'WITVEN2'}, 'price':
 {'N': '78'}, 'sku': {'S': 'F123'}}, {'quantity': {'N': '50'}, 'warehouse': {'S':
 'WITVEN3'}, 'price': {'N': '70'}, 'sku': {'S': 'F123'}}]
```

Hay varios enfoques para realizar actualizaciones por lotes en DynamoDB. El enfoque adecuado depende de factores como los requisitos de ACID o idempotencia, el número de elementos que se van a actualizar y el conocimiento de las API.

Operaciones masivas eficientes

Cuándo utilizar este patrón

Estos patrones son útiles para realizar eficientemente actualizaciones masivas en elementos de DynamoDB.

- El intérprete de comandos de DynamoDB no es compatible con el caso de uso de producción.
- `TransactWriteItems`: hasta 100 actualizaciones individuales con condiciones o sin ellas, ejecutándose como una agrupación ACID de todo o nada

  Desventaja: se consume rendimiento adicional, 2 WCU por 1 KB de escritura.
- PartiQL `BatchExecuteStatement`: hasta 25 actualizaciones con condiciones o sin ellas

  Desventaja: se requiere lógica adicional para distribuir las solicitudes en lotes de 25.
- AWS Step Functions: operaciones masivas con tasa limitada para desarrolladores familiarizados con AWS Lambda.

  Desventaja: el tiempo de ejecución es inversamente proporcional al límite de velocidad. Limitado por el tiempo de espera máximo de la función de Lambda. La funcionalidad implica que los cambios de datos que se producen entre la lectura y la escritura pueden sobrescribirse. Para obtener más información, consulte Backfilling an Amazon DynamoDB Time to Live attribute using Amazon EMR: Part 2.
- AWS Glue y Amazon EMR: operación masiva de tasa limitada con paralelismo administrado. Para aplicaciones o actualizaciones que no son urgentes, estas opciones pueden ejecutarse en

segundo plano, lo que solo consume un pequeño porcentaje del rendimiento. Ambos servicios utilizan el conector emr-dynamodb para realizar operaciones de DynamoDB. Estos servicios realizan una lectura grande seguida de una escritura grande de elementos actualizados con una opción de límite de velocidad.

Desventaja: el tiempo de ejecución es inversamente proporcional al límite de velocidad. La funcionalidad incluye que los cambios de datos que se producen entre la lectura y la escritura pueden sobrescribirse. No puede leer desde los índices secundarios globales (GSI). Consulte Backfilling an Amazon DynamoDB Time to Live attribute using Amazon EMR: Part 2.

- Interprete de comandos de DynamoDB: operaciones masivas con límite de velocidad mediante consultas similares a SQL. Puede leer los GSI para mejorar la eficiencia.

  Desventaja: el tiempo de ejecución es inversamente proporcional al límite de velocidad. Consulte Rate limited bulk operations in DynamoDB Shell.

Uso del patrón

Las actualizaciones masivas pueden tener implicaciones significativas en los costos, sobre todo si utiliza el modo de rendimiento bajo demanda. Hay un equilibrio entre velocidad y costos si utiliza el modo de rendimiento aprovisionado. Configurar el parámetro de límite de velocidad de forma muy estricta puede llevar a un tiempo de procesamiento muy largo. Puede determinar aproximadamente la velocidad de la actualización mediante el tamaño medio del elemento y el límite de velocidad.

De forma alternativa, puede determinar la cantidad de rendimiento necesaria para el proceso basándose en la duración esperada del proceso de actualización y el tamaño medio de los elementos. Las referencias de blog compartidas con cada patrón proporcionan detalles sobre la estrategia, la implementación y las limitaciones de utilizar el patrón. Para obtener más información, consulte Cost-effective bulk processing with Amazon DynamoDB.

Existen múltiples enfoques para realizar actualizaciones masivas en una tabla de DynamoDB activa. El enfoque adecuado depende de factores como los requisitos de ACID o idempotencia, el número de elementos que se van a actualizar y el conocimiento de las API. Es importante considerar el equilibrio entre costo y tiempo. La mayoría de los enfoques mencionados anteriormente ofrecen la opción de limitar el rendimiento utilizado por el trabajo de actualización masiva.

Prácticas recomendadas para implementar el control de versiones en DynamoDB

En sistemas distribuidos como DynamoDB, el control de versiones de elementos mediante bloqueo positivo evita actualizaciones conflictivas. Al realizar un seguimiento de las versiones de los elementos y utilizar escrituras condicionales, las aplicaciones pueden administrar modificaciones simultáneas, lo que garantiza la integridad de los datos en entornos de alta simultaneidad.

El bloqueo positivo es una estrategia que se utiliza para garantizar que las modificaciones de datos se apliquen correctamente y sin conflictos. En lugar de bloquear los datos cuando se leen (como en el bloqueo negativo), el bloqueo positivo comprueba si los datos han cambiado antes de volver a escribirlos. En DynamoDB, esto se logra a través de una forma de control de versiones, donde cada elemento incluye un identificador que se incrementa con cada actualización. Al actualizar un elemento, la operación solo se realizará correctamente si la coincidencia del identificador es la esperada por su aplicación.

Cuándo utilizar este patrón

Este patrón es útil en los siguientes casos:

- Varios usuarios o procesos pueden intentar actualizar el mismo elemento simultáneamente.
- Garantizar la integridad y la coherencia de los datos es primordial.
- Es necesario evitar la sobrecarga y la complejidad de administrar bloqueos distribuidos.

Entre los ejemplos se incluyen:

- Aplicaciones de comercio electrónico en las que los niveles de inventario se actualizan con frecuencia.
- Plataformas colaborativas en las que varios usuarios editan los mismos datos.
- Sistemas financieros en los que los registros de transacciones deben mantener la coherencia.

Desventajas

Aunque el bloqueo positivo y las comprobaciones condicionales proporcionan una sólida integridad de los datos, conllevan las siguientes desventajas:

Conflictos de simultaneidad

En entornos de alta simultaneidad, aumenta la probabilidad de conflictos, lo que puede provocar un mayor número de reintentos y mayores costos de escritura.

Complejidad de la implementación

Agregar control de versiones a elementos y controlar condiciones puede agregar complejidad a la lógica de la aplicación.

Sobrecarga de almacenamiento adicional

Almacenar los números de versión de cada elemento aumenta ligeramente los requisitos de almacenamiento.

Diseño de patrón

Para implementar este patrón, el esquema de DynamoDB debe incluir un atributo de versión para cada elemento. A continuación, mostramos un esquema sencillo:

- Clave de partición: un identificador único para cada elemento (por ejemplo, `ItemId`).
- Atributos:
  - `ItemId`: el identificador único del elemento.
  - `Version`: un número entero que representa el número de versión del elemento.
  - `QuantityLeft`: el inventario restante del elemento.

Cuando se crea un elemento por primera vez, el atributo `Version` se establece en 1. Con cada actualización, el número de versión aumenta en uno.

VersionControl

Primary key Partition key: ItemID	Attributes	
Bananas	Version	QuantityLeft
	1	10
Apples	Version	QuantityLeft
	1	5
Oranges	Version	QuantityLeft
	1	7

Uso del patrón

Para implementar este patrón, siga estos pasos en el flujo de la aplicación:

1. Lea la versión actual del elemento.

   Recupere el elemento actual de DynamoDB y lea su número de versión.

   ```
   def get_document(item_id):
       response = table.get_item(Key={'ItemID': item_id})
       return response['Item']

   document = get_document('Bananas')
   current_version = document['Version']
   ```

2. Incremente el número de versión en la lógica de la aplicación. Esta será la versión esperada para la actualización.

   ```
   new_version = current_version + 1
   ```

3. Intente actualizar el elemento mediante una expresión condicional para garantizar la coincidencia del número de versión.

   ```
   def update_document(item_id, qty_bought, current_version):
       try:
           response = table.update_item(
               Key={'ItemID': item_id},
               UpdateExpression="set #qty = :qty, Version = :v",
   ```

```
            ConditionExpression="Version = :expected_v",
            ExpressionAttributeNames={
                '#qty': 'QuantityLeft'
            },
            ExpressionAttributeValues={
                ':qty': qty_bought,
                ':v': current_version + 1,
                ':expected_v': current_version
            },
            ReturnValues="UPDATED_NEW"
        )
        return response
    except ClientError as e:
        if e.response['Error']['Code'] == 'ConditionalCheckFailedException':
            print("Update failed due to version conflict.")
        else:
            print("Unexpected error: %s" % e)
        return None

update_document('Bananas', 2, new_version)
```

Si la actualización se realiza correctamente, el valor de QuantityLeft del elemento se reducirá en dos.

VersionControl

Primary key Partition key: ItemID	Attributes	
Bananas	Version	QuantityLeft
	2	8
Apples	Version	QuantityLeft
	1	5
Oranges	Version	QuantityLeft
	1	7

4. Gestione los conflictos si se producen.

   Si se produce un conflicto (por ejemplo, otro proceso ha actualizado el elemento desde la última vez que lo leyó), gestione el conflicto de forma adecuada, por ejemplo, volviendo a intentar la operación o alertando al usuario.

Esto requerirá una lectura adicional del elemento para cada reintento, así que limite el número total de reintentos que permite antes de que falle completamente el bucle de solicitud.

```
def update_document_with_retry(item_id, new_data, retries=3):
    for attempt in range(retries):
        document = get_document(item_id)
        current_version = document['Version']

        result = update_document(item_id, qty_bought, current_version)

        if result is not None:
            print("Update succeeded.")
            return result
        else:
            print(f"Retrying update... ({attempt + 1}/{retries})")

    print("Update failed after maximum retries.")
    return None

update_document_with_retry('Bananas', 2)
```

Implementar el control de versiones de elementos mediante DynamoDB con bloqueo positivo y comprobaciones condicionales es un patrón eficaz para garantizar la integridad de los datos en aplicaciones distribuidas. Aunque implica cierta complejidad y posibles desventajas en cuanto al rendimiento, resulta inestimable en escenarios que requieren un control sólido de la simultaneidad. Mediante un diseño cuidadoso del esquema y la implementación de los controles necesarios en la lógica de la aplicación, puede administrar eficazmente las actualizaciones simultáneas y mantener la coherencia de datos.

Puede encontrar orientación y estrategias adicionales sobre cómo implementar el control de versiones de los datos de DynamoDB en el blog de bases de datos de AWS.

Prácticas recomendadas para interpretar los informes de facturación y uso de AWS en DynamoDB

En este documento se explican los códigos de facturación `UsageType` de los cargos relacionados con DynamoDB.

AWS proporciona informes de costos y uso (CUR) que contienen datos sobre los servicios utilizados. Puede utilizar AWS Cost and Usage Report para publicar informes de facturación en Amazon S3 en formato CSV. Al configurar el CUR, puede optar por desglosar los períodos de tiempo por hora, día o mes. Además, puede elegir si desea desglosar el uso por ID de recurso o no. Para obtener más información sobre cómo generar el CUR, consulte Creating Cost and Usage Reports.

En la exportación a CSV encontrará una lista de los atributos relevantes de cada línea. A continuación, se muestran ejemplos de posibles atributos:

- lineitem/UsageStartDate: fecha y hora de inicio de la partida en UTC, incluidas.
- lineitem/UsageEndDate: fecha y hora de fin de la partida correspondiente en UTC, no incluidas.
- lineitem/ProductCode: para DynamoDB es AmazonDynamoDB.
- lineitem/UsageType:código de descripción específico del tipo de uso, tal como se detalla en este documento.
- lineitem/Operation: nombre que da contexto al cargo, por ejemplo, el nombre de la operación que ha generado el cargo (opcional).
- lineitem/ResourceId: identificador del recurso que ha generado el uso. Está disponible si el CUR incluye un desglose por ID de recurso.
- lineitem/UsageAmount: cantidad de uso en la que se ha incurrido durante el periodo de tiempo especificado.
- lineitem/UnblendedCost: costo de dicho uso.
- lineitem/LineItemDescription: descripción textual de la partida.

Para obtener más información sobre el diccionario de datos de CUR, consulte Cost and Usage Report (CUR) 2.0. Tenga en cuenta que los nombres exactos varían según el contexto.

Un `UsageType` es una cadena con un valor como `ReadCapacityUnit-Hrs`, `USW2-ReadRequestUnits`, `EU-WriteCapacityUnit-Hrs` o `USE1-TimedPITRStorage-ByteHrs`. Cada tipo de uso comienza con un prefijo de región opcional. Si no está presente, equivale a la región us-east-1. Si está presente, consulte la siguiente tabla donde aparece el código de región de facturación abreviada asociado al código y nombre de región convencionales.

Por ejemplo, el uso denominado `USW2-ReadRequestUnits` hace referencia a las unidades de solicitud de lectura consumidas en us-west-2.

Código de región de facturación	Código de región	Nombre de la región
AFS1	af-south-1	África (Ciudad del Cabo)
APE1	ap-east-1	Asia-Pacífico (Hong Kong)
APN1	ap-northeast-1	Asia-Pacífico (Tokio)
APN2	ap-northeast-2	Asia-Pacífico (Seúl)
APN3	ap-northeast-3	Asia-Pacífico (Osaka)
APS1	ap-southeast-1	Asia-Pacífico (Singapur)
APS2	ap-southeast-2	Asia-Pacífico (Sídney)
APS3	ap-south-1	Asia-Pacífico (Mumbai)
APS4	ap-southeast-3	Asia-Pacífico (Yakarta)
APS5	ap-south-2	Asia-Pacífico (Hyderabad)
APS6	ap-southeast-4	Asia-Pacífico (Melbourne)
CAN1	ca-central-1	Canadá (centro)
UE	eu-west-1	Europa (Irlanda)
EUC1	eu-central-1	Europa (Fráncfort)
EUC2	eu-central-2	Europa (Zúrich)
EUN1	eu-north-1	Europa (Estocolmo)
EUS1	eu-south-1	Europa (Milán)
EUS2	eu-south-2	Europa (España)
EUW1	eu-west-1	Europa (Irlanda)
EUW2	eu-west-2	Europa (Londres)

Código de región de facturación	Código de región	Nombre de la región
EUW3	eu-west-3	Europa (París)
ILC1	Il-central-1	Israel (Tel Aviv)
MEC1	me-central-1	Medio Oriente (EAU)
MES1	me-south-1	Medio Oriente (Baréin)
SAE1	sa-east-1	América del Sur (São Paulo)
USE1 (predeterminado)	us-east-1	Este de EE. UU. (Norte de Virginia)
USE2	us-east-2	Este de EE. UU. (Ohio)
UGE1	us-gov-east-1	Gobierno del este de los EE. UU.
UGW1	us-gov-west-1	Gobierno del oeste de los EE. UU.
USW1	us-west-1	Oeste de EE. UU. (Norte de California)
USW2	us-west-2	Oeste de EE. UU. (Oregón)

En las siguientes secciones, utilizamos el patrón `REG-UsageType` para analizar los cargos de DynamoDB, donde REG equivale a la región en la que se ha producido el uso y UsageType es el código del tipo de cargo. Por ejemplo, si ve una partida para `USW1- ReadCapacityUnit-Hrs` en su archivo CSV, significa que se ha usado la capacidad de lectura aprovisionada en US-West-1. En ese caso, se denominaría `REG-ReadCapacityUnit-Hrs`.

Temas

- Capacidad de desempeño
- Transmisión
- Almacenamiento

- Copia de seguridad y restauración
- Transferencia de datos
- Información de colaboradores de Amazon CloudWatch
- DynamoDB Accelerator (DAX)

Capacidad de desempeño

Capacidad aprovisionada de lecturas y escrituras

Cuando se crea una tabla de DynamoDB en el modo de capacidad aprovisionada, debe especificar la capacidad de lectura y escritura que considera que va a necesitar su aplicación. El tipo de uso depende de la clase de tabla (acceso estándar o estándar de acceso poco frecuente). El aprovisionamiento de lectura y escritura se basa en la tasa de consumo por segundo, pero los cargos se cobran por hora en función de la capacidad aprovisionada.

UsageType	Unidades	Grado de detalle	Descripción
REG-ReadCapacityUnit-Hrs	Horas de RCU	Hora	Se cobra por las lecturas en el modo de capacidad aprovisionada con la clase de tabla estándar.
REG-IA-ReadCapacityUnit-Hrs	Horas de RCU	Hora	Se cobra por las lecturas en el modo de capacidad aprovisionada con la clase de tabla Standard-IA.
REG-WriteCapacityUnit-Hrs	Horas de WCU	Hora	Se cobra por las escrituras en el modo de capacidad aprovisionada con la clase de tabla estándar.

UsageType	Unidades	Grado de detalle	Descripción
REG-IA-WriteCapacityUnit-Hrs	Horas de WCU	Hora	Se cobra por las escrituras en el modo de capacidad aprovisionada con la clase de tabla Standard-IA.

Capacidad reservada de lecturas y escrituras

Con la capacidad reservada, se abona una tarifa inicial única y se adquiere el compromiso de utilizar un nivel mínimo aprovisionado durante un periodo concreto. La capacidad reservada se factura a una tarifa por hora con descuento. Cualquier capacidad que aprovisione que supere la capacidad reservada se cobrará de acuerdo con la tarifa de capacidad aprovisionada estándar. La capacidad reservada está disponible para las unidades de capacidad de lectura y escritura aprovisionadas de una sola región (RCU y WCU) en las tablas de DynamoDB que utilizan la clase de tabla estándar. Tanto la capacidad reservada de 1 año como la de 3 años se facturan con los mismos SKU.

UsageType	Unidades	Grado de detalle	Descripción
REG-Heavy Usage:dynamodb.read	Horas de RCU	Por adelantado y, luego, mensualmente	Cargos por lecturas de capacidad reservada: un cargo inicial único y un cargo mensual al principio de cada mes que cubre todas las horas de RCU comprometidas con descuento durante el mes. Contará con las partidas de REG-ReadCapacityUnit-Hrs sin costo correspondientes.

UsageType	Unidades	Grado de detalle	Descripción
REG-Heavy Usage:dynamodb.write	Horas de WCU	Por adelantado y, luego, mensualmente	Cargos por escrituras de capacidad reservada: un cargo inicial único y un cargo mensual al principio de cada mes que cubre todas las horas de WCU comprometidas con descuento durante el mes. Contará con las partidas de REG-WriteCapacityUnit-Hrs sin costo equivalentes.

Capacidad bajo demanda de lecturas y escrituras

Cuando se crea una tabla de DynamoDB en el modo de capacidad bajo demanda, solo se paga por las lecturas y escrituras que realiza la aplicación. Los precios de las solicitudes de lectura y escritura dependen de la clase de tabla.

UsageType	Unidades	Grado de detalle	Descripción
REG-ReadRequestUnits	RRU	Unidad	Se cobra por las lecturas en modo de capacidad bajo demanda con la clase de tabla estándar.
REG-IA-ReadRequestUnits	RRU	Unidad	Se cobra por las lecturas en modo de capacidad bajo

UsageType	Unidades	Grado de detalle	Descripción
			demanda con la clase de tabla Standard-IA.
REG-WriteRequestUnits	WRU	Unidad	Se cobra por las escrituras en modo de capacidad bajo demanda con la clase de tabla estándar.
REG-IA-WriteRequestUnits	WRU	Unidad	Se cobra por las escrituras en modo de capacidad bajo demanda con la clase de tabla Standard-IA.

Lecturas y escrituras de tablas globales

DynamoDB cobra por el uso de las tablas globales en función de los recursos utilizados en cada tabla de réplica. En el caso de las tablas globales aprovisionadas, las solicitudes de escritura de las tablas globales se miden en WCU replicadas (rWCU) en lugar de en WCU estándares. Además, las escrituras en los índices secundarios globales de las tablas globales se miden en WCU. En el caso de las tablas globales bajo demanda, las solicitudes de escritura se miden en WRU replicadas (rWRU) en lugar de en WRU estándares. La cantidad de rWCU o rWRU consumidas para la replicación depende de la versión de las tablas globales que utilice. El precio depende de la clase de tabla.

Las escrituras en índices secundarios globales (GSI) se facturan con unidades de escritura estándar (WCU y WRU). Las solicitudes de lectura y el almacenamiento de datos se facturan de forma idéntica a las tablas de una sola región.

Si agrega una réplica de tabla para crear o ampliar una tabla global en nuevas regiones, DynamoDB cobra por la restauración de una tabla en las regiones agregadas por gigabyte de los datos restaurados. Los datos restaurados se cobran como REG-RestoreDataSize-Bytes. Consulte Copia de seguridad y restauración para DynamoDB para obtener más información. La replicación entre regiones y la adición de réplicas a las tablas que contienen datos también conllevan gastos por la transferencia de datos.

Al seleccionar el modo de capacidad bajo demanda para las tablas globales de DynamoDB, solo paga por los recursos que utilice la aplicación en cada tabla de réplicas.

UsageType	Unidades	Grado de detalle	Descripción
REG-ReplWriteCapacityUnit-Hrs	Horas de rWCU	Hora	Tabla global, aprovisionada, clase de tabla estándar.
REG-IA-ReplWriteCapacityUnit-Hrs	Horas de rWCU	Hora	Tabla global, aprovisionada, clase de tabla Standard-IA.
REG-ReplWriteRequestUnits	rWRU	Unidad	Tabla global, bajo demanda, clase de tabla estándar.
REG-IA-ReplWriteRequestUnits	rWRU	Unidad	Tabla global, bajo demanda, clase de tabla Standard-IA.

Transmisión

DynamoDB cuenta con dos tecnologías de transmisión: DynamoDB Streams y Kinesis. Cada una tiene un precio diferente.

DynamoDB Streams cobra por leer datos en unidades de solicitud de lectura. Cada llamada a la API `GetRecords` se factura como una solicitud de lectura de secuencias. No se cobran las llamadas a la API `GetRecords` realizadas por AWS Lambda como parte de los desencadenadores de DynamoDB ni por las tablas globales de DynamoDB como parte de la replicación.

UsageType	Unidades	Grado de detalle	Descripción
REG-Streams-RequestsCount	Recuento	Unidad	Unidades de solicitud de lectura de DynamoDB Streams.

Amazon Kinesis Data Streams cobra por unidades de captura de datos de cambios. DynamoDB cobra una unidad de captura de datos de cambios por cada escritura (hasta 1 KB). Para los elementos de más de 1 KB se requieren unidades de captura de datos de cambios adicionales. Solo se paga por las escrituras que realice la aplicación sin gestionar la capacidad de rendimiento en la tabla.

UsageType	Unidades	Grado de detalle	Descripción
REG-ChangeDataCaptureUnits-Kinesis	Unidades de CDC	Unidad	Unidades de captura de datos de cambios para Kinesis Data Streams.

Almacenamiento

DynamoDB mide el tamaño de los datos facturables al agregar el tamaño de byte sin procesar de los datos y una capacidad de almacenamiento por elemento que depende de las características que haya habilitado.

> **Note**
>
> Cuando se utilice `DescribeTable`, los valores de uso de almacenamiento en el CUR serán más altos en comparación con los valores de almacenamiento, ya que `DescribeTable` no incluye la sobrecarga de almacenamiento por elemento.

El almacenamiento se calcula por hora, pero el precio mensual se calcula sobre un promedio de los cargos por hora.

Aunque el almacenamiento `UsageType` utiliza `ByteHrs` como sufijo, el uso del almacenamiento en el CUR se mide en GB y se cobra por GB al mes.

UsageType	Unidades	Grado de detalle	Descripción
REG-TimedStorage-ByteHrs	GB	Mes	Cantidad de almacenamiento que

UsageType	Unidades	Grado de detalle	Descripción
			utilizan las tablas e índices de DynamoDB para las tablas de la clase estándar.
REG-IA-TimedStorage- ByteHrs	GB	Mes	Cantidad de almacenamiento que utilizan las tablas e índices de DynamoDB para las tablas de la clase Standard-IA.

Copia de seguridad y restauración

DynamoDB ofrece dos tipos de copias de seguridad: copias de seguridad de recuperación en un momento dado (PITR) y copias de seguridad bajo demanda. Los usuarios también pueden realizar restauraciones a partir de esas copias de seguridad en tablas de DynamoDB. Los cargos que figuran a continuación se refieren tanto a las copias de seguridad como a las restauraciones.

Los cargos por almacenamiento de copias de seguridad se cobran el primer día del mes y se realizan ajustes a lo largo del mes a medida que se agregan o eliminan copias de seguridad. Lea la publicación del blog Understanding Amazon DynamoDB On-demand Backups and Billing para obtener más información.

UsageType	Unidades	Grado de detalle	Descripción
REG-Timed BackupStorage-Byte Hrs	GB	Mes	Almacenamiento consumido por las copias de seguridad bajo demanda de las tablas de DynamoDB y los índices secundarios locales.

UsageType	Unidades	Grado de detalle	Descripción
TimedPITRStorage-ByteHrs	GB	Mes	Almacenamiento utilizado por las copias de seguridad de recuperación en un momento dado (PITR). DynamoDB monitorea el tamaño de las tablas activadas para PITR de forma continua durante todo el mes para determinar los cargos de las copias de seguridad y factura por el almacenamiento, siempre y cuando la opción PITR esté activada.
REG-RestoreDataSize-Bytes	GB	Tamaño	Tamaño total de los datos restaurados (incluidos los datos de tablas, los índices secundarios locales y los índices secundarios globales) medidos en GB a partir de las copias de seguridad de DynamoDB.

AWS Backup

AWS Backup es un servicio de copia de seguridad completamente administrado que facilita la centralización y automatización de las copias de seguridad de datos en servicios de AWS en la

nube y en las instalaciones. AWS Backup se cobra en función del almacenamiento (almacenamiento en frío o en caliente), las actividades de restauración y la transferencia de datos entre regiones. Los siguientes cargos de `UsageType` aparecen en el ProductCode AWSBackup y no en AmazonDynamoDB.

UsageType	Unidades	Grado de detalle	Descripción
REG-WarmStorage-ByteHrs-DynamoDB	GB	Mes	El almacenamiento utilizado por las copias de seguridad de DynamoDB se gestiona según el rendimiento de AWS Backup a lo largo del mes y se mide en GB al mes.
REG-CrossRegion-WarmBytes-DynamoDB	GB	Tamaño	Datos transferidos a una región de AWS diferente, ya sea dentro de la misma cuenta o a una cuenta de AWS distinta. Cuando se copian las copias de seguridad de una región a otra se cobran cargos por transferencias entre regiones. El cargo siempre se factura a la cuenta desde la que se transfieren los datos.
REG-Restore-WarmBytes-DynamoDB	GB	Tamaño	Tamaño total de los datos restaurados desde un almacenam

UsageType	Unidades	Grado de detalle	Descripción
			iento en caliente, medido en GB.
REG-ColdStorage-ByteHrs-DynamoDB	GB	Mes	El almacenamiento en frío utilizado por las copias de seguridad de DynamoDB se gestiona según el rendimiento de AWS Backup a lo largo del mes y se mide en GB al mes.
REG-Restore-ColdBytes-DynamoDB	GB	Mes	Tamaño total de los datos restaurados desde un almacenamiento en frío, medido en GB.

Exportación e importación

Puede exportar datos desde DynamoDB a Amazon S3 o importar datos desde Amazon S3 a una nueva tabla de DynamoDB.

Aunque el `UsageType` usa `Bytes` como sufijo, el uso de la exportación y la importación en el CUR se mide y se cobra por GB.

UsageType	Unidades	Grado de detalle	Descripción
REG-ExportDataSize-Bytes	GB	Tamaño	Cargo por exportar datos a S3. DynamoDB cobra por los datos exportados en función del tamaño de la tabla básica de DynamoDB

UsageType	Unidades	Grado de detalle	Descripción
			(datos de la tabla e índices secundarios locales) en el momento específico en que se crea la exportación.
REG-ImportDataSize-Bytes	GB	Tamaño	Cargo por la importación de datos desde S3. El tamaño se calcula en función del tamaño del objeto sin comprimir de los datos de Amazon S3. La importación a tablas con GSI no conlleva cargos adicionales.
REG-IncrementalExportDataSize-Bytes	GB	Tamaño	Cargo por el tamaño de los datos procesados a partir de la copia de seguridad continua para producir exportaciones incrementales.

Transferencia de datos

La actividad de transferencia de datos puede aparecer asociada al servicio de DynamoDB. DynamoDB no cobra por la transferencia de datos entrantes ni por los datos transferidos entre DynamoDB y otros servicios de AWS en la misma región de AWS (es decir, 0,00 USD por GB). Los datos transferidos entre regiones de AWS (por ejemplo, entre DynamoDB en la región Este de EE. UU. (Norte de Virginia) y Amazon EC2 en la región Europa [Irlanda]) se cobran en ambos lados de la transferencia.

UsageType	Unidades	Grado de detalle	Descripción
REG-DataTransfer-In-Bytes	GB	Unidades	Datos transferidos desde Internet a DynamoDB.
REG-DataTransfer-Out-Bytes	GB	Unidades	Datos transferidos a Internet desde DynamoDB.

Información de colaboradores de Amazon CloudWatch

CloudWatch Contributor Insights para DynamoDB es una herramienta de diagnóstico que permite identificar las claves de acceso más frecuente y sometidas a más limitaciones en la tabla de DynamoDB. Los siguientes cargos de `UsageType` aparecen en el ProductCode AmazonCloudWatch y no en AmazonDynamoDB.

UsageType	Unidades	Grado de detalle	Descripción
REG-CW:ContributorEventsManaged	Eventos procesados	Unidades	Cantidad de eventos de DynamoDB procesados. Por ejemplo, en una tabla que tenga CloudWatch Contributor Insights activado, cada vez que se lee o escribe un elemento, se cuenta como un evento. Si la tabla tiene una clave de clasificación, se cobran dos eventos.
REG-CW:ContributorRulesManaged	Recuento de reglas	Mes	DynamoDB crea reglas para identific

UsageType	Unidades	Grado de detalle	Descripción
			ar los elementos a los que se accede con más frecuenci a y las claves más limitadas al activar CloudWatch Contribut or Insights. Este cargo se aplica a las reglas agregadas para cada entidad (tablas y GSI) configuradas para registrar la informaci ón de los colaborad ores de CloudWatch.

DynamoDB Accelerator (DAX)

DynamoDB Accelerator (DAX) se factura por hora en función del tipo de instancia seleccionada para el servicio. Los siguientes cargos hacen referencia a las instancias de DynamoDB Accelerator aprovisionadas. Los siguientes cargos de `UsageType` aparecen en el ProductCode AmazonDAX y no en AmazonDynamoDB.

UsageType	Unidades	Grado de detalle	Descripción
REG-NodeU sage:dax- <INSTANCETYPE>	Horas de nodo	Hora	Uso por hora de un tipo de instancia concreto. El precio es por hora de nodo consumida, desde el momento en que se lanza un nodo hasta que se cierra. Cada hora parcial de nodo consumida se

UsageType	Unidades	Grado de detalle	Descripción
			facturará como hora completa. DAX cobra por cada nodo de un clúster de DAX. Si tiene un clúster con varios nodos, verá varias partidas en el informe de facturación.

El tipo de instancia será uno de los valores de la siguiente lista. Para obtener más información acerca de los tipos de nodos, consulte Nodos.

- r3.2xlarge, r4.8xlarge o r5.8xlarge
- r3.4xlarge, r4.large o r5.large
- r3.8xlarge, r4.xlarge o r5.xlarge
- r3.2xlarge, r5.12xlarge o t2.medium
- r3.4xlarge, r4.large o r5.large
- r3.xlarge, r5.16xlarge o t2.small
- r4.16xlarge, r5.24xlarge o t3.medium
- r4.2xlarge, r5.2xlarge o t3.small
- r4.4xlarge o r5.4xlarge

Migración de una tabla de DynamoDB de una cuenta a otra

Puede migrar una tabla de Amazon DynamoDB de una cuenta a otra para implementar una estrategia de varias cuentas o una estrategia de copias de seguridad. También puede hacerlo por motivos de prueba, depuración o conformidad. Un caso de uso común es copiar tablas de DynamoDB en los entornos de producción, provisional, pruebas y desarrollo, en los que se utiliza una cuenta de AWS diferente en cada uno.

DynamoDB ofrece dos opciones para migrar tablas de una cuenta de AWS a otra:

- AWS Backup para realizar copias de seguridad y restauraciones entre cuentas: AWS Backup es un servicio de copia de seguridad totalmente administrado que le permite administrar de forma centralizada las copias de seguridad de varios servicios de AWS. Con su funcionalidad de copia de seguridad y restauración entre cuentas, puede hacer una copia de seguridad de una tabla de DynamoDB en una cuenta y restaurarla en otra cuenta de la misma AWS Organization.
- Exportación e importación de DynamoDB a Amazon S3: las características de exportación e importación de DynamoDB a Amazon S3 le permiten realizar una exportación completa a un bucket de Amazon S3 y, a continuación, importar esos datos a una nueva tabla de otra cuenta de AWS. Este enfoque es adecuado cuando necesita realizar migraciones entre cuentas que no forman parte de la misma AWS Organization o si no quiere utilizar AWS Backup.

> **Note**
>
> La importación desde Amazon S3 no admite tablas con índices secundarios locales (LSI), pero sí admite índices secundarios globales (GSI). Para obtener más información sobre LSI y GSI, consulte Mejora del acceso con índices secundarios en DynamoDB.

Temas

- Migración de una tabla mediante AWS Backup para la copia de seguridad y la restauración entre cuentas
- Migración de una tabla mediante la exportación a S3 y la importación desde S3

Migración de una tabla mediante AWS Backup para la copia de seguridad y la restauración entre cuentas

Requisitos previos

- Las cuentas de AWS de origen y destino debe pertenecer a la misma organización en el servicio de AWS Organizations
- Permisos de AWS Identity and Access Management (IAM) válidos para crear y utilizar almacenes de AWS Backup

Para obtener más información sobre la configuración de copias de seguridad entre cuentas, consulte Creating backup copies across AWS accounts.

Información sobre precios

AWS cobra por la copia de seguridad (en función del tamaño de la tabla), por cualquier copia de datos entre regiones de AWS (en función de la cantidad de datos), por la restauración (en función de la cantidad de datos) y por cualquier cargo de almacenamiento continuo. Para evitar cargos continuos, puede eliminar la copia de seguridad si no la necesita después de la restauración.

Para obtener más información acerca de los precios, consulte Precios de AWS Backup.

Paso 1: activar las características avanzadas para DynamoDB y copia de seguridad entre cuentas

1. Tanto en la cuenta de AWS de origen como en la de destino, acceda a la consola de administración de AWS y abra la consola de AWS Backup.
2. Seleccione la opción Configuración.
3. En Características avanzadas para copias de seguridad de Amazon DynamoDB, confirme que la opción Características avanzadas esté activada. Si no lo está, seleccione Activar.
4. En Administración entre cuentas, en Copia de seguridad entre cuentas, elija Activar.

Paso 2: crear un almacén de copias de seguridad en la cuenta de origen y la cuenta de destino

1. En las cuentas de AWS de origen, abra la consola de AWS Backup.
2. Elija Almacenes de Backup.
3. Seleccione Create Backup vault (Crear almacén de copias de seguridad).
4. Copie y guarde el nombre de recurso de Amazon (ARN) de los almacenes de copia de seguridad creados y de la cuenta de AWS de destino.
5. Necesitará los ARN de los dos almacenes de copias de seguridad (el de origen y el de destino), al copiar la copia de seguridad de la tabla de DynamoDB de una cuenta a otra.

Paso 3: crear una copia de seguridad de la tabla de DynamoDB en la cuenta de origen

1. En la Página del Panel de AWS Backup, seleccione Crear copia de seguridad bajo demanda.
2. En la sección Configuración, seleccione DynamoDB como Tipo de recurso y, a continuación, seleccione el nombre de la tabla.

3. En la lista desplegable Almacén de copias de seguridad, seleccione el almacén de copias de seguridad que creó en la cuenta de origen.
4. Seleccione el Periodo de retención deseado.
5. Seleccione Create on-demand backup (Crear copia de seguridad bajo demanda).
6. Monitorice el estado del trabajo de copia de seguridad en la pestaña Trabajos de copia de seguridad de la página Trabajos de AWS Backup.

Paso 4: copiar la copia de seguridad de la tabla de DynamoDB de la cuenta de origen a la cuenta de destino

1. Una vez finalizado el trabajo de copia de seguridad, abra la consola de AWS Backup en la cuenta de origen y seleccione Almacenes de copia de seguridad.
2. En Backups (Copias de seguridad), seleccione la copia de seguridad de la tabla de DynamoDB. Elija Acciones y, a continuación, elija Copiar.
3. Especifique la región de AWS de la cuenta de destino.
4. Para el ARN del almacén externo, escriba el ARN del almacén de copias de seguridad que creó en la cuenta de destino.
5. En el almacén de copias de seguridad de la cuenta de destino, habilite el acceso de una cuenta de origen para poder copiar las copia de seguridad.

Paso 5: restaurar la copia de seguridad de la tabla de DynamoDB en la cuenta de destino

1. En la cuenta de AWS de destino, abra la consola de AWS Backup y elija Almacenes de copia de seguridad
2. En Copias de seguridad, seleccione la copia de seguridad que ha copiado de la cuenta de origen. En Acciones, elija Restaurar.
3. Introduzca el nombre para la nueva tabla de DynamoDB, el cifrado que tendrá esta nueva tabla, la clave con la que desea que se cifre la restauración y cualquier otra opción.
4. Cuando se complete la restauración, se indicará que el estado de la tabla es Activo.

Migración de una tabla mediante la exportación a S3 y la importación desde S3

Requisitos previos

- Debe habilitar la recuperación en un momento dado (PITR) en su tabla para poder realizar la exportación a S3. Para obtener más información, consulte Habilitación de la recuperación en un momento dado en DynamoDB.
- Permisos de IAM válidos para realizar la exportación. Para obtener más información, consulte Solicitar una exportación de tabla en DynamoDB.
- Permisos de IAM válidos suficientes para realizar la importación. Para obtener más información, consulte Solicitud de una importación de tabla en DynamoDB.

Información sobre precios

AWS cobra por PITR (en función del tamaño de la tabla y del tiempo durante el que se habilite PITR). Si no necesita PITR salvo para la exportación, puede desactivarla una vez que termine. AWSTambién cobra por las solicitudes realizadas en S3, por almacenar los datos exportados en S3 y por la importación (en función del tamaño sin comprimir de los datos importados).

Para obtener más información sobre los precios de DynamoDB, consulte Precios de DynamoDB.

> **Note**
>
> Hay límites en cuanto al tamaño y la cantidad de objetos al importar de S3 a DynamoDB. Para obtener más información, consulte Cuotas de importación.

Solicitud de una exportación de tabla a Amazon S3

1. Inicie sesión en la consola de administración de AWS y abra la consola de DynamoDB.
2. En el panel de navegación del lado izquierdo de la consola, elija Exports to S3 (Exportación a S3).
3. Elija una tabla de origen y un bucket de S3 de destino. Introduzca la URL del bucket de la cuenta de destino con el formato `s3://bucketname/prefix`. `/prefix` es una carpeta opcional que lo ayuda a mantener organizado el bucket de destino.

4. Seleccione Exportación completa. Una exportación completa genera la instantánea completa de la tabla en el momento que especifique.
   a. Seleccione Hora actual para exportar la instantánea completa de la tabla más reciente.
   b. En Formato de archivo exportado, elija entre DynamoDB JSON y Amazon Ion. La opción predeterminada es DynamoDB JSON.
5. Haga clic en el botón Export (Exportar) para comenzar la exportación.
6. Las exportaciones de tablas pequeñas deberían completarse en unos minutos, pero las tablas que tienen terabytes pueden tardar más de una hora.

Solicitud de la importación de una tabla desde Amazon S3

1. Inicie sesión en la consola de administración de AWS y abra la consola de DynamoDB.
2. En el panel de navegación del lado izquierdo de la consola, elija Import from S3 (Importar desde S3).
3. En la página que aparece, seleccione Import from S3 (Importar desde S3).
4. Introduzca la URL de origen de Amazon S3. También puede encontrarla si utiliza el botón Buscar S3. La ruta esperada tiene el formato `s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/data/`.
5. Especifique si es S3 bucket owner (Propietario del bucket de S3).
6. En Compresión de archivos de importación, seleccione GZIP para que coincida con la de la exportación.
7. En Formato de archivo de importación, seleccione DynamoDB JSON para que coincida con el de la exportación.
8. Seleccione Siguiente. En Especificar los detalles de la tabla, elija las opciones de la nueva tabla que se creará para almacenar los datos.
9. Seleccione Siguiente. En Configurar la configuración de la tabla, personalice cualquier ajuste adicional de la tabla, si procede.
10. Vuelva a seleccionar Next (Siguiente) para revisar las opciones de importación y, a continuación, haga clic en Import (Importar) para iniciar la tarea de importación. Verá la nueva tabla en Importaciones desde S3 con el estado Importando. No puede acceder a la tabla durante este tiempo. Las importaciones pequeñas deberían completarse en unos minutos, pero las tablas que tienen terabytes pueden tardar más de una hora.

11. Una vez completada la importación, el estado se muestra como Activo y puede empezar a utilizar la tabla.

Mantenimiento de las tablas sincronizadas durante la migración

Si puede pausar las operaciones de escritura en la tabla de origen durante la migración, el origen y la salida deberían coincidir exactamente después de la migración. Si no puede pausar las operaciones de escritura, lo normal es que la tabla de destino se quede un poco por detrás de la de origen tras la migración. Para poner al día la tabla de origen, puede utilizar flujos (DynamoDB Streams o Kinesis Data Streams para DynamoDB) para reproducir las escrituras que se produjeron en la tabla de origen desde la copia de seguridad o la exportación.

Debería empezar leyendo los registros de flujo antes de la fecha y hora en que exportó la tabla de origen a S3. Por ejemplo, si la exportación a S3 se produjo a las 14:00 horas y la importación a la tabla de destino concluyó a las 23:00 horas, deberá iniciar la lectura del flujo de DynamoDB a las 13:58 horas. La tabla de opciones de flujo para la captura de datos de cambios resume las características de cada modelo de flujo.

El uso de DynamoDB Streams con Lambda ofrece un enfoque simplificado para sincronizar los datos entre las tablas de DynamoDB de origen y destino. Puede utilizar una función de Lambda para reproducir cada escritura de la tabla de destino.

> **Note**
>
> Los elementos se mantienen en DynamoDB Streams durante 24 horas, por lo que debe planificar la realización de su copia de seguridad y restauración o la exportación e importación dentro de ese periodo de tiempo.

Recomendaciones para integrar DAX con las aplicaciones de DynamoDB

DynamoDB Accelerator (DAX) es un servicio de almacenamiento en caché compatible con DynamoDB que proporciona un rendimiento en memoria rápido para aplicaciones exigentes, como las que tienen mucho tráfico de lectura. Con DAX, puede lograr tiempos de respuesta en microsegundos para acceder a los datos que se solicitan con frecuencia. En estas recomendaciones

de DynamoDB Accelerator, se proporciona información exhaustiva y las prácticas recomendadas para integrar DAX con sus aplicaciones de DynamoDB.

Estas recomendaciones proporcionan conocimientos básicos para aquellos que utilicen por primera vez DAX o que quieran optimizar sus configuraciones existentes. En ellas se tratan varios temas como, por ejemplo, cuándo usar DAX y cómo crear un clúster de DAX. También se incluyen ejemplos prácticos y explicaciones detalladas para ayudarle a implementar DAX de forma efectiva en sus proyectos. Por último, las recomendaciones ofrecen estrategias avanzadas que debe implementar para maximizar las capacidades de almacenamiento en caché de DAX y garantizar que las aplicaciones sean rápidas y escalables.

Temas

- Evaluación de la idoneidad de DAX para sus casos de uso
- Configuración del cliente de DAX
- Configuración del clúster de DAX
- Dimensionamiento del clúster de DAX
- Implementación de un clúster
- Administración de las operaciones del clúster
- Monitoreo de DAX

Evaluación de la idoneidad de DAX para sus casos de uso

En esta sección, se explica cuándo y por qué usar DAX. Estas recomendaciones le ayudarán a determinar si la integración de DAX con DynamoDB es ventajosa para los patrones de carga de trabajo, los requisitos de rendimiento y las necesidades de coherencia de datos de su aplicación. También cubre situaciones en las que DAX podría no ser adecuado, por ejemplo, para cargas de trabajo con mucho tráfico de escritura y con datos a los que se accede con poca frecuencia.

En esta sección

- Cuándo y por qué elegir DAX
- Cuándo no utilizar DAX

Cuándo y por qué elegir DAX

Puede considerar la posibilidad de agregar DAX a su pila de aplicaciones en varios escenarios. Por ejemplo, utilice DAX para reducir la latencia general de las solicitudes de lectura en DynamoDB o

para minimizar las lecturas repetidas de los mismos datos de una tabla. La siguiente lista contiene ejemplos de escenarios en los que puede beneficiarle integrar DAX con DynamoDB:

- Se necesita un alto rendimiento
  - Lecturas de baja latencia: debería considerar la posibilidad de utilizar DAX si su aplicación requiere tiempos de respuesta de microsegundos para lograr lecturas coherentes posteriores. DAX también puede reducir drásticamente el tiempo de respuesta para acceder a los datos que se leen con frecuencia.
- Cargas de trabajo con mucho tráfico de lectura
  - Aplicaciones con mucho tráfico de lectura: en el caso de las aplicaciones con una alta relación de lectura y escritura, por ejemplo, de 10:1 o más, DAX genera más aciertos de caché y menos datos obsoletos. Esto reduce las lecturas en una tabla. Para evitar leer datos obsoletos de la caché si la aplicación tiene un elevado tráfico de escritura, asegúrese de establecer un valor de Uso del período de vida (TTL) en DynamoDB más bajo para la caché.
  - Almacenamiento en caché de consultas comunes: si su aplicación lee con frecuencia los mismos datos (por ejemplo, productos populares en una plataforma de comercio electrónico), DAX puede atender estas solicitudes directamente desde su caché.
- Patrones de tráfico en ráfagas
  - Escalado de tablas más fluido: DAX ayuda a suavizar la repercusión de los picos de tráfico repentinos. DAX proporciona un búfer para escalar verticalmente con fluidez la capacidad de las tablas de DynamoDB, lo que reduce el riesgo de que se produzcan limitaciones en la lectura.
  - Mayor rendimiento de lectura para cada elemento: DynamoDB asigna particiones individuales para cada elemento. Sin embargo, una partición comienza a limitar las lecturas de un elemento cuando alcanza las 3000 unidades de capacidad de lectura (RCU). DAX permite escalar las lecturas de un solo elemento a más de 3000 RCU.
- Optimización de costos
  - Reducción de los costos de DynamoDB: la lectura de DAX puede reducir las lecturas enviadas a una tabla de DynamoDB, lo que puede repercutir directamente en los costos. Con una alta tasa de aciertos de caché, el costo reducido de lectura de tablas puede superar el costo de un clúster de DAX, lo que se traduce en una reducción de los costos netos.
- Requisitos de coherencia de datos
  - Coherencia final: DAX admite lecturas coherentes posteriores. Esto hace que DAX sea adecuado para casos de uso en los que la coherencia inmediata no es fundamental.

- Almacenamiento en caché de escritura indirecta: las escrituras que se realizan en DAX son indirectas Una vez que DAX confirma que ha escrito un elemento en DynamoDB, mantiene esa versión del elemento en la caché de elementos. Este mecanismo de escritura indirecta ayuda a mantener una mayor coherencia de datos entre la caché y la base de datos, pero utiliza recursos adicionales de clúster de DAX.

Cuándo no utilizar DAX

Aunque DAX es potente, no es adecuado para todos los escenarios. La siguiente lista contiene ejemplos de escenarios en los que la integración de DAX con DynamoDB no es adecuada:

- Cargas de trabajo con mucho tráfico de escritura: la principal ventaja de DAX es que acelera las lecturas, pero las escrituras utilizan más recursos de DAX que las lecturas. Si su aplicación tiene sobre todo mucho tráfico de escritura, las ventajas de DAX podrían ser limitadas.
- Datos que se leen con poca frecuencia: si su aplicación accede a los datos con poca frecuencia o a una amplia gama de datos que raramente se reutilizan (datos fríos), es probable que tenga una cache hit ratio baja. En este caso, es posible que la sobrecarga que supone el mantenimiento de la memoria caché no justifique las mejoras de rendimiento.
- Lecturas o escrituras en bloque: si su aplicación realiza más escrituras en bloque que escrituras individuales, deberá escribir de forma directa en DAX. Además, para las lecturas en bloque, debe realizar análisis de tabla completos directamente en una tabla de DynamoDB.
- Requisitos de transacciones o coherencia estrictos: DAX transfiere las lecturas altamente coherentes y las llamadas de TransactGetItems a una tabla de DynamoDB. Debe realizar estas lecturas de forma directa en el clúster de DAX para evitar el uso de los recursos del clúster. Los elementos que se lean de esta manera no se almacenarán en caché; por lo tanto, enrutarlos a través de DAX no sirve para nada.
- Aplicaciones sencillas con requisitos de rendimiento modestos: en el caso de aplicaciones que tienen requisitos de rendimiento modestos y tolerancia a la latencia directa de DynamoDB, es posible que no sea necesario asumir la complejidad y el costo de agregar DAX. DynamoDB gestiona por sí solo el alto rendimiento y proporciona un rendimiento de milisegundos de un solo dígito.
- Necesidades de consulta complejas más allá del acceso clave-valor: DAX está optimizado para patrones de acceso clave-valor. Si su aplicación requiere capacidades de consulta complejas con filtrado complejo, como las operaciones Query y Scan, las ventajas del almacenamiento en caché de DAX podrían ser limitadas.

En estas situaciones, utilice Amazon ElastiCache (Redis OSS) como alternativa. ElastiCache (Redis OSS) admite estructuras de datos avanzadas, como listas, conjuntos y hash. También ofrece características, como pub/sub, índices geoespaciales y scripts.

- Requisitos de conformidad: actualmente, DAX no ofrece las mismas acreditaciones de conformidad que DynamoDB. Por ejemplo, DAX aún no ha obtenido la acreditación de SOC.

Configuración del cliente de DAX

El clúster de DAX es un clúster basado en instancias al que se puede acceder mediante varios SDK de DAX. Cada SDK proporciona a los desarrolladores opciones configurables, como requestTimeout y conexiones, para cumplir con los requisitos específicos de la aplicación.

Al configurar el cliente de DAX, una consideración fundamental es la escala de la aplicación cliente, en concreto, la proporción de instancias de cliente con respecto a las instancias de servidor de DAX (que tiene un máximo de 11). Las grandes flotas de instancias de clientes pueden generar numerosas conexiones a instancias de servidores de DAX, lo que podría sobrecargarlas. En esta guía se describen las prácticas recomendadas para la configuración de clientes de DAX.

Prácticas recomendadas

1. Instancias de clientes: implemente instancias de clientes únicas para garantizar la reutilización de las instancias en todas las solicitudes. Para conocer los detalles de implementación, consulte the section called “Paso 4: ejecutar un ejemplo de aplicación”.
2. Tiempos de espera de solicitudes: aunque las aplicaciones suelen requerir tiempos de espera de solicitud bajos para garantizar una latencia mínima en los sistemas ascendentes, establecer tiempos de espera demasiado bajos puede causar problemas. Los tiempos de espera bajos pueden provocar una reconexión frecuente a las instancias del servidor cuando los servidores de DAX experimentan picos de latencia temporales. Cuando se agota el tiempo de espera, el cliente de DAX finaliza la conexión del nodo del servidor existente y establece una nueva. Como el establecimiento de la conexión requiere muchos recursos, numerosas conexiones consecutivas pueden sobrecargar los servidores de DAX. Le recomendamos lo siguiente:
   - Mantenimiento de la configuración predeterminada de tiempo de espera de las solicitudes.
   - Si es necesario reducir los tiempos de espera, implemente subprocesos de aplicación independientes con valores de tiempo de espera más bajos e incluya mecanismos de reintento con un retraso exponencial.

3. Tiempo de espera de conexión: para la mayoría de las aplicaciones, recomendamos mantener la configuración de tiempo de espera de conexión predeterminada.
4. Conexiones simultáneas: determinados SDK, como JavaV2, permiten ajustar las conexiones simultáneas al servidor de DAX. Consideraciones clave:
   - Las instancias del servidor de DAX pueden gestionar hasta 40 000 conexiones simultáneas.
   - La configuración predeterminada es adecuada para la mayoría de los casos de uso.
   - Las instancias de clientes grandes combinadas con un alto número de conexiones simultáneas pueden sobrecargar los servidores.
   - Los valores de conexión simultánea más bajos reducen el riesgo de sobrecarga del servidor.
   - Ejemplo de cálculo del rendimiento:
     - Al suponer una latencia de solicitud de 1 ms, en teoría, cada conexión puede gestionar 1000 solicitudes por segundo.
     - Para un clúster de 3 nodos, una sola instancia de cliente que se conecte a todos los nodos puede procesar 3000 solicitudes por segundo.
     - Con 10 conexiones, el cliente puede gestionar aproximadamente 30 000 solicitudes por segundo.

       Recomendación: comience con una configuración de conexión simultánea más baja y valide mediante pruebas de rendimiento con los patrones de carga de trabajo de producción esperados.

Configuración del clúster de DAX

El clúster de DAX es un clúster administrado, pero puede ajustar sus configuraciones para adaptarlas a los requisitos de su aplicación. Dada su estrecha integración con las operaciones de la API de DynamoDB, debe tener en cuenta los siguientes aspectos a la hora de integrar su aplicación con DAX.

En esta sección

- Precios de DAX
- Caché de elementos y caché de consultas
- Selección de la configuración de TTL para las cachés
- Almacenamiento en caché de varias tablas con un clúster de DAX
- Replicación de datos en tablas globales de DAX y DynamoDB

- Disponibilidad por región de DAX
- Comportamiento del almacenamiento en caché en DAX

Precios de DAX

El costo de un clúster depende de la cantidad y el tamaño de los nodos que haya aprovisionado. A cada nodo se le factura por cada hora de ejecución en el clúster. Para obtener más información, consulte los precios de Amazon DynamoDB.

Los aciertos de caché no incurren en costos de DynamoDB, pero afectan a los recursos del clúster de DAX. Los errores de caché generan costos de lectura de DynamoDB y requieren recursos de DAX. Las escrituras generan costos de escritura de DynamoDB y afectan a los recursos del clúster de DAX para realizar la escritura mediante proxy.

Caché de elementos y caché de consultas

DAX mantiene una caché de elementos y una caché de consultas. Comprender las diferencias entre estas cachés puede ayudarle a determinar las características de rendimiento y coherencia que ofrecen a su aplicación.

Características de caché	Caché de elementos	Caché de consultas
Finalidad	Almacena los resultados de las operaciones de las API GetItem y BatchGetItem.	Almacena los resultados de las operaciones de las API Query y Scan. Estas operaciones pueden devolver varios elementos en función de las condiciones de la consulta y no de claves de elementos específicas.
Tipo de acceso	Utiliza el acceso basado en claves. Cuando una aplicación solicita datos mediante `GetItem` o `BatchGetItem` , DAX comprueba primero la caché	Utiliza el acceso basado en parámetros. DAX almacena en caché el conjunto de resultados y las operaciones de API `Query` y `Scan`. DAX atiende

Características de caché	Caché de elementos	Caché de consultas
	de elementos utilizando la clave principal de los elementos solicitados. Si el elemento se encuentra en caché y no ha caducado, DAX se lo devuelve inmediatamente sin acceder a la tabla de DynamoDB.	las solicitudes posteriores con los mismos parámetros que incluyen las mismas condiciones de consulta, tabla e índice, desde la caché. Esto reduce considerablemente los tiempos de respuesta y el consumo de rendimiento de lectura de DynamoDB.
Invalidación de caché	DAX replica automáticamente los elementos actualizados en la caché de elementos de los nodos del clúster de DAX en los siguientes casos: • Escribe una actualización del elemento a través de la caché. • Lee una versión actualizada del elemento desde la tabla.	La caché de consultas es más difícil de invalidar que la caché de elementos. Es posible que las actualizaciones de los elementos no se asignen directamente a las consultas o análisis en caché. Debe ajustar cuidadosamente el TTL de la caché de consultas para mantener la coherencia de datos. Las escrituras indirectas en DAX o la tabla base no se reflejan en la caché de consultas hasta que caduca el TTL de la respuesta previamente almacenada en caché y DAX realiza una nueva consulta en DynamoDB.

Características de caché	Caché de elementos	Caché de consultas
Índice secundario global	Dado que la operación de la API `GetItem` no se admite en índices secundarios locales ni en índices secundari os globales, la caché de elementos solo almacena en caché las lecturas de la tabla base.	La caché de consultas almacena en caché las consultas realizadas en tablas e índices.

Selección de la configuración de TTL para las cachés

TTL determina el periodo durante el cual los datos se almacenan en la caché antes de que queden obsoletos. Después de este tiempo, los datos se actualizan automáticamente en la siguiente solicitud. La elección de la configuración de TTL adecuada para sus cachés de DAX implica un equilibrio entre la optimización del rendimiento de la aplicación y la coherencia de los datos. Como no existe una configuración de TTL universal que funcione para todas las aplicaciones, la configuración de TTL óptima varía en función de las características y los requisitos específicos de la aplicación. Le recomendamos que empiece por una configuración de TTL conservadora siguiendo estas recomendaciones. A continuación, ajuste la configuración de TTL de forma iterativa en función de los datos y la información sobre el rendimiento de la aplicación.

DAX mantiene una lista de elementos menos usados recientemente (LRU) para la caché de elementos. La lista de LRU registra cuándo se escribieron los elementos por primera vez o se leyeron por última vez de la memoria caché. Cuando la memoria del nodo de DAX está llena, DAX expulsa los elementos más antiguos aunque aún no hayan caducado para dejar sitio a los nuevos. El algoritmo de LRU siempre está habilitado y el usuario no puede configurarlo.

Para establecer una duración TTL adecuada para sus aplicaciones, tenga en cuenta lo siguiente:

Comprensión de los patrones de acceso a los datos

- Cargas de trabajo con mucho tráfico de lectura: en el caso de las aplicaciones con cargas de trabajo con mucho tráfico de lectura y actualizaciones de datos poco frecuentes, establezca una duración de TTL más larga para reducir el número de errores de caché. Una duración de TTL mayor también reduce la necesidad de acceder a la tabla de DynamoDB subyacente.

- Cargas de trabajo con mucho tráfico de escritura: en el caso de las aplicaciones con actualizaciones frecuentes que no se escriben de forma indirecta en DAX, establezca una duración de TTL más corta para garantizar que la memoria caché se mantenga coherente con la base de datos. Una duración más corta de TTL también reduce el riesgo de proporcionar datos obsoletos.

Evaluación de los requisitos de rendimiento de su aplicación

- Sensibilidad a la latencia: si su aplicación requiere una latencia baja más que datos recientes, utilice una duración de TTL más larga. Una duración de TTL más larga maximiza los aciertos de caché, lo que reduce la latencia media de lectura.
- Rendimiento y escalabilidad: una mayor duración de TTL reduce la carga en las tablas de DynamoDB y mejora el rendimiento y la escalabilidad. Sin embargo, debe equilibrar esto con la necesidad de tener datos actualizados.

Analice la expulsión de la caché y el uso de la memoria

- Límites de memoria caché: monitorice el uso de memoria de su clúster de DAX. Una duración más larga de TTL podría hacer que se almacenaran más datos en la caché, lo que podría alcanzar los límites de memoria y provocar expulsiones basadas en el LRU.

Uso de métricas y monitorización para ajustar el TTL

Revise periódicamente las métricas, por ejemplo, los aciertos y errores de caché y el uso de la CPU y la memoria. Ajuste la configuración de TTL en función de estas métricas para encontrar un equilibrio óptimo entre el rendimiento y el nivel de actualización de los datos. Si los errores de caché son altos y el uso de memoria es bajo, aumente la duración de TTL para aumentar la tasa de aciertos de caché.

Tenga en cuenta los requisitos empresariales y el cumplimiento

Las políticas de retención de datos pueden dictar la duración máxima de TTL que puede establecer para la información confidencial o personal.

Comportamiento de la caché si establece TTL en cero

Si establece TTL en 0, la caché de elementos y la caché de consultas tienen los siguientes comportamientos:

- Caché de elementos: los elementos de la caché se actualizan solo cuando se produce una expulsión de LRU o una operación de escritura indirecta.
- Caché de consultas: las respuestas a las consultas no se almacenan en caché.

Almacenamiento en caché de varias tablas con un clúster de DAX

Para las cargas de trabajo que tienen varias tablas pequeñas de DynamoDB que no necesitan cachés individuales, un único clúster de DAX almacena en caché las solicitudes de estas tablas. Esto proporciona un uso más flexible y eficiente de DAX, especialmente para las aplicaciones que acceden a varias tablas y requieren lecturas de alto rendimiento.

Al igual que las API del plano de datos de DynamoDB, las solicitudes de DAX requieren un nombre de tabla. Si usa varias tablas en el mismo clúster de DAX, no necesita ninguna configuración específica. Sin embargo, debe asegurarse de que los permisos de seguridad del clúster permitan el acceso a todas las tablas almacenadas en caché.

Consideraciones sobre el uso de DAX con varias tablas

Cuando utilice DAX con varias tablas de DynamoDB, debe tener en cuenta lo siguiente:

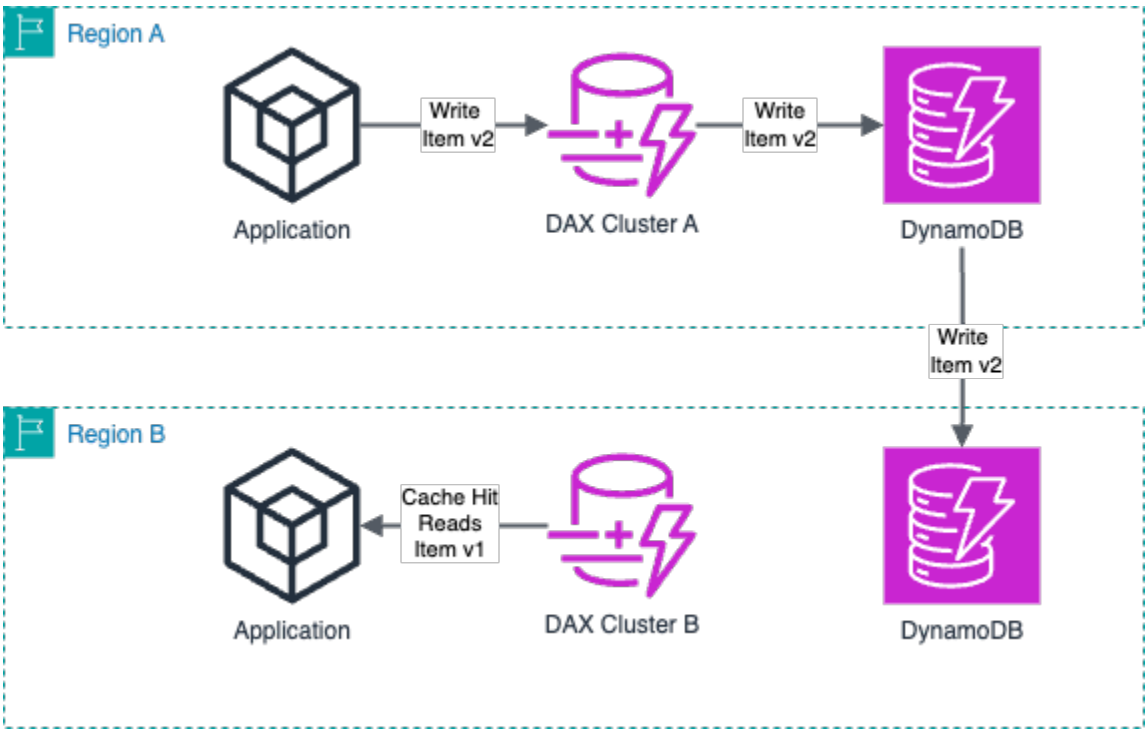
- Administración de memoria: cuando utilice DAX con varias tablas, debe tener en cuenta el tamaño total del conjunto de datos de trabajo. Todas las tablas del conjunto de datos compartirán el mismo espacio de memoria del tipo de nodo que haya seleccionado.
- Asignación de recursos: los recursos del clúster de DAX se comparten entre todas las tablas almacenadas en caché. Sin embargo, una tabla con mucho tráfico puede provocar la expulsión de datos de las tablas más pequeñas vecinas.
- Economías de escala: agrupe los recursos más pequeños en un clúster de DAX más grande para hacer una media el tráfico y seguir un patrón más estable. Para la cantidad total de recursos de lectura que requiere el clúster de DAX, también resulta económico tener tres o más nodos. Esto también aumenta la disponibilidad de todas las tablas almacenadas en caché en el clúster.

Replicación de datos en tablas globales de DAX y DynamoDB

DAX es un servicio basado en regiones, por lo que un clúster solo ve el tráfico que hay en su Región de AWS. Las tablas globales escriben alrededor de la caché cuando replican datos de otra región.

Una duración más larga de TTL puede provocar que los datos obsoletos permanezcan en la región secundaria durante más tiempo que en la región principal. Esto puede provocar errores de caché en la caché local de la región secundaria.

El siguiente diagrama muestra la replicación de datos que se produce en las tablas globales de la región A de origen. El clúster de DAX de la región B no ve inmediatamente los datos recién replicados de la región A de origen.



Disponibilidad por región de DAX

No todas las regiones que admiten tablas de DynamoDB admiten la implementación de clústeres de DAX. Si su aplicación requiere una latencia de lectura baja a través de DAX, consulte primero la lista de regiones que admiten DAX. A continuación, seleccione una región para la tabla de DynamoDB.

Comportamiento del almacenamiento en caché en DAX

DAX realiza el almacenamiento en caché negativo y de metadatos. Conocer estos comportamientos de almacenamiento en caché le ayudará a administrar eficazmente los metadatos de los atributos de los elementos almacenados en caché y las entradas de caché negativas.

- Almacenamiento en caché de metadatos: los clústeres de DAX conservan de forma indefinida metadatos sobre los nombres de atributo de los elementos almacenados en caché. Estos metadatos se conservan incluso después de que el elemento caduque o se expulse de la caché.

Con el tiempo, las aplicaciones que utilizan un número ilimitado de nombres de atributos pueden provocar el agotamiento de la memoria en el clúster de DAX. Esta limitación se aplica solo a los nombres de atributos de nivel superior, pero no a los nombres de atributo anidados. Algunos ejemplos de nombres de atributos ilimitados son las marcas temporales, los UUID y los ID de sesión. Aunque puede utilizar las marcas de tiempo y los ID de sesión como valores de atributos, le recomendamos que utilice nombres de atributos más cortos y predecibles.

- Almacenamiento en caché negativo: si se produce un error en la caché y la lectura de una tabla de DynamoDB no proporciona ningún elemento que coincida, DAX agrega una entrada de caché negativa en la caché de elementos o consultas correspondiente. Esta entrada permanece hasta que caduque la duración de TTL de la caché o se produzca una escritura indirecta. DAX sigue devolviendo esta entrada de caché negativa para futuras solicitudes.

  Si el comportamiento negativo del almacenamiento en caché no se ajusta al patrón de su aplicación, lea directamente la tabla de DynamoDB cuando DAX devuelva un resultado vacío. También le recomendamos que establezca una duración de TTL de la caché más baja para evitar que haya resultados vacíos durante mucho tiempo en la caché y mejorar la coherencia con la tabla.

Dimensionamiento del clúster de DAX

La capacidad y disponibilidad totales de un clúster de DAX dependen del tipo y el número de nodos. Más nodos en el clúster aumentan su capacidad de lectura, pero no la capacidad de escritura. Los tipos de nodos más grandes (hasta r5.8xlarge) pueden gestionar más escrituras, pero tener muy pocos nodos puede afectar a la disponibilidad si se produce un error en el nodo. Para obtener más información acerca del dimensionamiento de su clúster de DAX, consulte Guía de tamaño del clúster de DAX.

En las siguientes secciones, se analizan los diferentes aspectos del dimensionamiento que se deben tener en cuenta para equilibrar el tipo y la cantidad de nodos a fin de crear un clúster escalable y rentable.

En esta sección

- Planificación de la disponibilidad
- Planificación del rendimiento de escritura
- Planificación del rendimiento de lectura
- Planificación del tamaño del conjunto de datos

- Cálculo de los requisitos aproximados de capacidad del clúster
- Estimación de la capacidad de rendimiento del clúster por tipo de nodo
- Escalado de la capacidad de escritura en los clústeres de DAX

Planificación de la disponibilidad

Al dimensionar un clúster de DAX, primero debe centrarse en la disponibilidad prevista. La disponibilidad de un servicio en clúster, como DAX, es una dimensión del número total de nodos del clúster. Dado que un clúster de un solo nodo no tiene tolerancia a errores, su disponibilidad es la misma que la de un nodo. En un clúster de 10 nodos, la pérdida de un solo nodo tiene una repercusión mínima en la capacidad general del clúster. Esta pérdida no afecta directamente a la disponibilidad, ya que los nodos restantes pueden satisfacer las solicitudes de lectura. Para reanudar las escrituras, DAX nombra rápidamente un nuevo nodo principal.

DAX está basado en VPC. Utiliza un grupo de subredes para determinar en qué zonas de disponibilidad puede ejecutar nodos y qué direcciones IP utilizar desde las subredes. Para las cargas de trabajo de producción, se recomienda encarecidamente utilizar DAX con al menos tres nodos en zonas de disponibilidad diferentes. Esto garantiza que al clúster le quede más de un nodo para gestionar las solicitudes, incluso si se produce un error en un único nodo o zona de disponibilidad. Un clúster puede tener hasta 11 nodos, de los que uno es un nodo principal y 10 son réplicas de lectura.

Planificación del rendimiento de escritura

Todos los clústeres de DAX tienen un nodo principal para las solicitudes de escritura indirecta. El tamaño del tipo de nodo del clúster determina su capacidad de escritura. Agregar réplicas de lectura adicionales no aumenta la capacidad de escritura del clúster. Por lo tanto, debe tener en cuenta la capacidad de escritura durante la creación del clúster, ya que no podrá cambiar el tipo de nodo más adelante.

Si su aplicación necesita escribir de forma indirecta en DAX para actualizar la caché de elementos, considere la posibilidad de aumentar el uso de los recursos del clúster para facilitar la escritura. Las escrituras en DAX consumen aproximadamente 25 veces más recursos que las lecturas de aciertos de caché. Esto puede requerir un tipo de nodo más grande que el de los clústeres de solo lectura.

Para obtener información adicional sobre cómo determinar si la escritura indirecta o la escritura directa funcionan mejor para su aplicación, consulte Estrategias de escritura.

Planificación del rendimiento de lectura

La capacidad de lectura de un clúster de DAX depende de la proporción de aciertos de caché de la carga de trabajo. Dado que DAX lee los datos de DynamoDB cuando se produce un error de caché, consume aproximadamente 10 veces más recursos del clúster que un acierto de caché. Para aumentar los aciertos de caché, aumente la configuración de TTL de la caché para definir el periodo durante el que se almacena un elemento en la caché. Sin embargo, una mayor duración de TTL aumenta la probabilidad de leer versiones anteriores de los elementos, a menos que las actualizaciones se escriban de forma indirecta en DAX.

Para asegurarse de que el clúster tiene una capacidad de lectura suficiente, escale el clúster horizontalmente como se indica en Escalado horizontal de un clúster. Al agregar más nodos, se agregan réplicas de lectura al conjunto de recursos, mientras que al eliminar nodos se reduce la capacidad de lectura. Al seleccionar la cantidad de nodos y sus tamaños para un clúster, tenga en cuenta la cantidad mínima y máxima de capacidad de lectura necesaria. Si no puede escalar horizontalmente un clúster con tipos de nodos más pequeños para satisfacer sus requisitos de lectura, use un tipo de nodo más grande.

Planificación del tamaño del conjunto de datos

Cada tipo de nodo disponible tiene un tamaño de memoria establecido para que DAX almacene los datos en caché. Si un tipo de nodo es demasiado pequeño, el conjunto de datos de trabajo que solicita una aplicación no cabrá en la memoria y provocará errores de caché. Como los nodos más grandes admiten cachés más grandes, utilice un tipo de nodo mayor que el conjunto de datos estimado que necesita almacenar en caché. Una caché más grande también mejora la tasa de aciertos de caché.

Es posible que obtenga un rendimiento menor si guarda elementos en caché con pocas lecturas repetidas. Calcule el tamaño de la memoria de los elementos a los que se accede con frecuencia y asegúrese de que la caché sea lo suficientemente grande como para almacenar ese conjunto de datos.

Cálculo de los requisitos aproximados de capacidad del clúster

Puede calcular las necesidades de capacidad total de su carga de trabajo para ayudarle a seleccionar el tamaño y la cantidad adecuados de nodos del clúster. Para realizar esta estimación, calcule la variable solicitud normalizada por segundo (RPS normalizado). Esta variable representa el total de unidades de trabajo que la aplicación necesita que admita el clúster de DAX, incluidos los

aciertos de caché, los errores de caché y las escrituras. Para calcular el RPS normalizado, utilice los siguientes datos:

- `ReadRPS_CacheHit`: especifica el número de lecturas por segundo que provocan un acierto de caché.
- `ReadRPS_CacheMiss`: especifica el número de lecturas por segundo que provocan un error de caché.
- `WriteRPS`: especifica el número de escrituras por segundo que se realizarán de forma indirecta en DAX.
- `DaxNodeCount`: especifica el número de nodos del clúster de DAX.
- `Size`: especifica el tamaño del elemento que se va a escribir o leer en KB redondeado al KB más cercano.
- `10x_ReadMissFactor`: representa un valor de 10. Cuando se produce un error de caché, DAX utiliza aproximadamente 10 veces más recursos que los aciertos de caché.
- `25x_WriteFactor`: representa un valor de 25 porque una escritura indirecta de DAX utiliza aproximadamente 25 veces más recursos que los aciertos de caché.

Con la siguiente fórmula, puede calcular el RPS normalizado.

```
Normalized RPS = (ReadRPS_CacheHit * Size) + (ReadRPS_CacheMiss * Size *
 10x_ReadMissFactor) + (WriteRequestRate * 25x_WriteFactor * Size * DaxNodeCount)
```

Por ejemplo, observe los siguientes valores de entrada:

- `ReadRPS_CacheHit` = 50 000
- `ReadRPS_CacheMiss` = 1000
- `ReadMissFactor` = 1
- `Size` = 2 KB
- `WriteRPS` = 100
- `WriteFactor` = 1
- `DaxNodeCount` = 3

Si sustituye estos valores en la fórmula, puede calcular el RPS normalizado de la siguiente manera.

```
Normalized RPS = (50,000 Cache Hits/Sec * 2KB) + (1,000 Cache Misses/Sec * 2KB * 10) +
 (100 Writes/Sec * 25 * 2KB * 3)
```

En este ejemplo, el valor calculado del RPS normalizado es 135 000. Sin embargo, este valor de RPS normalizado no tiene en cuenta el mantenimiento del uso del clúster por debajo del 100 % ni los errores de nodos. Le recomendamos que agregue capacidad adicional. Para ello, determine el mayor de los dos factores multiplicadores: el objetivo de utilización o la tolerancia a los errores de nodos. A continuación, multiplique el RPS normalizado por el factor mayor para obtener un objetivo de solicitudes por segundo (RPS objetivo).

- Objetivo de utilización

  Dado que la repercusión en el rendimiento aumenta los errores de caché, no recomendamos ejecutar el clúster de DAX con un uso del 100 %. Lo ideal es mantener la utilización del clúster en un 70 % o por debajo de ese porcentaje. Para lograrlo, multiplique el RPS normalizado por 1,43.
- Tolerancia a errores de nodos

  Si un nodo falla, la aplicación debe poder distribuir sus solicitudes entre los nodos restantes. Para asegurarse de que los nodos se mantengan por debajo del 100 % de utilización, elija un tipo de nodo lo suficientemente grande como para absorber el tráfico adicional hasta que el nodo que ha fallado vuelva a estar en línea. En el caso de un clúster con menos nodos, cada nodo debe tolerar mayores aumentos de tráfico cuando se produce un error en uno de ellos.

  En el caso de que falle el nodo principal, DAX realiza automáticamente una conmutación por error a una réplica de lectura y la designa como nuevo nodo principal. Si falla un nodo de réplica, otros nodos del clúster de DAX pueden seguir atendiendo las solicitudes hasta que se consigue recuperar el nodo que ha fallado.

  Por ejemplo, un clúster de DAX de 3 nodos con un nodo fallido requiere un 50 % adicional de capacidad en los dos nodos restantes. Esto requiere un factor multiplicador de 1,5. Por el contrario, un clúster de 11 nodos con un nodo fallido requiere un 10 % de capacidad adicional en los nodos restantes o un factor de multiplicación de 1,1.

Con la siguiente fórmula, puede calcular el objetivo de RPS.

```
Target RPS = Normalized RPS * CEILING(TargetUtilization, NodeLossTolerance)
```

Por ejemplo, para calcular el objetivo de RPS, tenga en cuenta los siguientes valores:

- `Normalized RPS` = 135 000
- `TargetUtilization` = 1,43

  Como nuestro objetivo es una utilización máxima de los clústeres del 70 %, hemos configurado `TargetUtilization` en 1,43.
- `NodeLossTolerance` = 1,5

  Supongamos que estamos usando un clúster de 3 nodos, por lo tanto, hemos configurado `NodeLossTolerance` en una capacidad del 50 %.

Si sustituye estos valores en la fórmula, puede calcular el objetivo de RPS de la siguiente manera.

```
Target RPS = 135,000 * CEILING(1.43, 1.5)
```

En este ejemplo, dado que el valor de `NodeLossTolerance` es mayor que `TargetUtilization`, calculamos el valor del objetivo de RPS con `NodeLossTolerance`. Esto nos da un objetivo de RPS de 202 500, que es la cantidad total de capacidad que debe admitir el clúster de DAX. Para determinar la cantidad de nodos que necesitará en un clúster, asigne el objetivo de RPS a una columna adecuada de la siguiente tabla. Para este ejemplo de un objetivo de RPS de 202 500, necesitará el tipo de nodo dax.r5.large con tres nodos.

Estimación de la capacidad de rendimiento del clúster por tipo de nodo

Con la Target RPS formula, puede calcular la capacidad del clúster para diferentes tipos de nodos. En la siguiente tabla, se muestran las capacidades aproximadas de los clústeres con 1, 3, 5 y 11 tipos de nodos. Estas capacidades no sustituyen la necesidad de realizar pruebas de carga de DAX con sus propios patrones de solicitudes y datos. Además, estas capacidades no incluyen las instancias de tipo t, porque no tienen capacidad de CPU fija. La unidad de todos los valores de la siguiente tabla es el RPS normalizado.

Tipo de nodo (memoria)	1 nodo	3 nodos	5 nodos	11 nodos
dax.r5.24xlarge (768 GB)	1M	3 millones	5 millones	11 millones

Tipo de nodo (memoria)	1 nodo	3 nodos	5 nodos	11 nodos
dax.r5.16xlarge (512 GB)	1M	3 millones	5 millones	11 millones
dax.r5.12xlarge (384 GB)	1M	3 millones	5 millones	11 millones
dax.r5.8xlarge (256 GB)	1M	3 millones	5 millones	11 millones
dax.r5.4xlarge (128 GB)	600 000	1,8 millones	3 millones	6,6 millones
dax.r5.2xlarge (64 GB)	300 000	900 000	1,5 millones	3,3 millones
dax.r5.xlarge (32 GB)	150 000	450 000	750 000	1,65 millones
dax.r5.large (16 GB)	75 000	225 000	375 000	825 000

Debido al límite máximo de 1 millón de NPS (operaciones de red por segundo) de cada nodo, los nodos de tipo dax.r5.8xlarge o superiores no aportan capacidad de clúster adicional. Es posible que los tipos de nodos más grandes que 8xlarge no contribuyan a la capacidad de rendimiento total del clúster. Sin embargo, estos tipos de nodos pueden resultar útiles para almacenar un conjunto de datos de trabajo más grande en la memoria.

Escalado de la capacidad de escritura en los clústeres de DAX

Cada escritura en DAX consume 25 solicitudes normalizadas en cada nodo. Como hay un límite de 1 millón de RPS para cada nodo, un clúster de DAX está limitado a 40 000 escrituras por segundo, sin tener en cuenta el uso de lectura.

Si su caso de uso requiere más de 40 000 escrituras por segundo en la caché, debe utilizar clústeres de DAX independientes y particionar las escrituras entre ellos. Al igual que en DynamoDB, puede

aplicar un hash a la clave de partición de los datos que va a escribir en la caché. A continuación, utilice el módulo para determinar en qué partición escribir los datos.

En el siguiente ejemplo, se calcula el hash de una cadena de entrada. A continuación, se calcula el módulo del valor hash con 10.

```
def hash_modulo(input_string):
    # Compute the hash of the input string
    hash_value = hash(input_string)

    # Compute the modulus of the hash value with 10
    bucket_number = hash_value % 10

    return bucket_number

#Example usage
if _name_ == "_main_":
    input_string = input("Enter a string: ")
    result = hash_modulo(input_string)
    print(f"The hash modulo 10 of '{input_string}' is: {result}.")
```

Implementación de un clúster

La creación de un nuevo clúster de DAX requiere más configuraciones que las que se necesitan para DynamoDB. Estas configuraciones son especialmente para las redes, porque DAX se basa en Amazon VPC. Esto le ofrece un control total sobre su entorno de redes virtuales, incluida la ubicación de los recursos, la conectividad y la seguridad. En esta sección, se examinan las prácticas recomendadas para la configuración necesaria durante la creación del clúster.

Para obtener más información sobre la elección de los nodos de clúster, consulte Dimensionamiento del clúster de DAX.

En esta sección

- Configuración de las redes
- Configuración de la seguridad
- Grupo de parámetros
- Periodo de mantenimiento

Configuración de las redes

DAX utiliza un grupo de subredes para determinar en qué zonas de disponibilidad puede ejecutar nodos y qué direcciones IP utilizar de las subredes. Para minimizar la latencia entre la aplicación y DAX, las subredes y las zonas de disponibilidad de los servidores de aplicaciones y del clúster de DAX deben ser las mismas.

Le recomendamos distribuir los nodos de DAX entre varias zonas de disponibilidad. Con la opción predeterminada de asignación automática, puede hacerlo de forma automática.

Para obtener información sobre las prácticas recomendadas sobre la configuración de su VPC, consulte Introducción a Amazon VPC en la Guía del usuario de Amazon VPC.

Configuración de la seguridad

En esta sección, se describen las medidas de seguridad que debe implementar para las aplicaciones que utilizan DAX. En esta sección, también se analiza brevemente la compatibilidad de DAX con el cifrado de datos.

IAM

DAX y DynamoDB tienen mecanismos distintos de control de acceso. DAX requiere un rol de IAM para acceder a las tablas de DynamoDB. Este rol debe seguir el principio del privilegio mínimo y conceder acceso únicamente a tablas y operaciones de DynamoDB específicas, como GetItem y PutItem. Para obtener más información sobre los mecanismos de control de acceso proporcionados por DAX, consulte Control de acceso a DAX.

Cifrado

El cifrado en reposo y el cifrado en tránsito se configura al crear un clúster de DAX. De forma predeterminada, están habilitados. Le recomendamos que utilice la configuración de cifrado predeterminada a menos que los requisitos de la empresa lo impidan. Para obtener más información, consulte Cifrado en reposo de DAX y Cifrado en tránsito de DAX.

Grupo de parámetros

DAX aplica un conjunto de configuraciones a cada nodo de un clúster denominado grupo de parámetros. Puede cambiar esta configuración después de crear el clúster.

El grupo de parámetros de DAX contiene la configuración de TTL para la caché de elementos y la caché de consultas. La duración predeterminada de TTL es de 5 minutos. Puede anular la duración de TTL para cambiarla por cualquier valor entero superior o igual a 1 milisegundo.

No se pueden modificar los grupos de parámetros si una instancia de DAX en ejecución los está utilizando. Puede cambiar los valores de los grupos de parámetros durante el tiempo de inactividad de un clúster de DAX.

Periodo de mantenimiento

Para poder realizar actualizaciones y parches de software ocasionales en sus nodos, se configura un periodo de mantenimiento semanal para el clúster de DAX. Durante este periodo, DAX realiza actualizaciones continuas en los nodos. Los clústeres con más de un nodo no pierden la disponibilidad del clúster durante estas actualizaciones, pero tienen una capacidad menor hasta que el nodo regrese. Si su organización tiene un periodo predecible de bajo uso, considere la posibilidad de configurar el periodo de mantenimiento manualmente a ese momento.

Administración de las operaciones del clúster

DAX se encarga del mantenimiento y el estado del clúster automáticamente. Sin embargo, debe proporcionar información operativa para escalar el clúster horizontal o verticalmente para adaptarlo a sus patrones de uso. En esta sección, se describe el proceso recomendado para escalar los clústeres de DAX.

En esta sección

- Escalado horizontal de un clúster
- Escalado vertical de un clúster

Escalado horizontal de un clúster

Escalar un clúster de DAX implica ajustar su capacidad para satisfacer las demandas de rendimiento. Este ajuste se realiza aumentando o disminuyendo la cantidad de nodos (réplicas) del clúster mientras está en ejecución. Este proceso, conocido como escalado horizontal, ayuda a distribuir la carga de trabajo entre más nodos o a consolidarla en menos nodos cuando la demanda es baja.

Puede reducir y escalar horizontalmente el clúster de DAX mediante los comandos `decrease-replication-factor` o `increase-replication-factor` de la AWS CLI.

Aumento del factor de replicación (escalado horizontal)

Al aumentar el factor de replicación de un clúster de DAX, se agregan más nodos al clúster. En el siguiente ejemplo, se muestran los resultados del comando `increase-replication-factor`.

```
aws dax increase-replication-factor \
    --cluster-name yourClusterName  \
    --new-replication-factor desiredReplicationFactor
```

- En este comando, el argumento `cluster-name` especifica el nombre del clúster. Por ejemplo, *suNombreCluster*.
- El argumento `new-replication-factor` especifica el número total de nodos que se van a agregar en el cluster tras el escalado. Esto incluye el nodo principal y los nodos de réplica. Por ejemplo, si su clúster tiene actualmente 3 nodos y desea agregar 2 nodos más, defina el valor de `new-replication-factor` en 5.

Reducción del factor de replicación (reducción horizontal)

Al reducir el factor de replicación de un clúster de DAX, se eliminan los nodos del clúster. La eliminación de nodos puede ayudar a reducir los costos durante los periodos de baja demanda. En el siguiente ejemplo, se muestran los resultados del comando `decrease-replication-factor`.

```
aws dax decrease-replication-factor \
    --cluster-name yourClusterName  \
    --new-replication-factor desiredReplicationFactor
```

- En este comando, el argumento `cluster-name` especifica el nombre del clúster. Por ejemplo, *suNombreCluster*.
- El argumento `new-replication-factor` especifica el número reducido de nodos del clúster después del escalado. Este número debe ser inferior al factor de replicación actual y debe incluir el nodo principal. Por ejemplo, si su clúster tiene 5 nodos y desea eliminar 2 nodos, defina el valor de `new-replication-factor` en 3.

Consideraciones sobre el escalado horizontal

Tenga en cuenta lo siguiente cuando planifique el escalado horizontal:

- Nodo principal: el clúster de DAX incluye un nodo principal. El factor de replicación incluye este nodo principal. Por ejemplo, un factor de replicación de 3 significa que hay un nodo principal y dos nodos de réplica.
- Disponibilidad: agregar o eliminar nodos de DAX cambia la disponibilidad y la tolerancia a errores del clúster. Más nodos pueden mejorar la disponibilidad, pero también aumentan los costos.

- Migración de datos: al aumentar el factor de replicación, DAX gestiona automáticamente la distribución de los datos en el nuevo conjunto de nodos. Cuando un nodo nuevo comienza a atender el tráfico, su caché ya está preparada. Sin embargo, durante este proceso, es posible que se produzca un impacto temporal en el rendimiento durante la migración de datos.

Asegúrese de monitorizar atentamente los clústeres de DAX durante y después del proceso de escalado para asegurarse de que funcionan según lo esperado y realice los ajustes adicionales que sean necesarios.

Escalado vertical de un clúster

Para escalar verticalmente el tamaño de los nodos de un clúster existente, debe crear un clúster nuevo y migrar el tráfico de las aplicaciones al nuevo clúster. La migración a un nuevo clúster con nodos diferentes implica varios pasos para garantizar una transición fluida con el impacto mínimo en el rendimiento y la disponibilidad de la aplicación.

Para crear un nuevo clúster para escalar el tamaño del nodo verticalmente, tenga en cuenta lo siguiente:

- Acceda a su configuración actual: revise las métricas de su clúster de DAX actual para determinar el tamaño y la cantidad de nodos nuevos que necesita. Utilice esta información como entrada para definir el tamaño del clúster. Para obtener más información, consulte Dimensionamiento del clúster de DAX.
- Configure un nuevo clúster de DAX: cree un nuevo clúster de DAX con el tipo y la cantidad de nodos que haya determinado. Puede utilizar los ajustes de configuración existentes de su grupo de parámetros, a menos que necesite realizar ajustes.
- Sincronice los datos: dado que DAX es una capa de almacenamiento en caché para DynamoDB, no es necesario migrar los datos directamente. Sin embargo, el nuevo clúster de DAX no tendrá ningún conjunto de datos de trabajo en la memoria hasta que no le envíe tráfico.
- Actualice la configuración de la aplicación: actualice la configuración de la aplicación para que apunte al nuevo punto de conexión del clúster de DAX. Es posible que necesite cambiar el código o actualizar las variables de entorno, en función de la configuración de la aplicación.

  Para reducir el impacto que se produce al cambiar a un clúster nuevo, envíe tráfico canario al nuevo clúster desde una pequeña parte de su flota de aplicaciones. Para ello, puede implementar lentamente las actualizaciones de las aplicaciones o utilizar una entrada de DNS de enrutamiento basada en la ponderación enfrente de su punto de conexión de DAX.

- Monitorice y optimice: después de cambiar al nuevo clúster de DAX, monitorice atentamente sus métricas y registros de rendimiento para detectar cualquier problema. Prepárese para ajustar la cantidad de nodos en función de los patrones de carga de trabajo actualizados.

  Hasta que el nuevo clúster almacene en caché su conjunto de datos de trabajo correctamente, verá tasas de error de caché y latencias más altas.
- Retire del servicio el clúster antiguo: cuando esté seguro de que el nuevo clúster funciona según lo esperado, retire el clúster de DAX antiguo de forma segura para evitar costos innecesarios.

Monitoreo de DAX

Puede monitorizar las métricas clave, por ejemplo, la tasa de aciertos de caché, para garantizar un rendimiento óptimo del clúster de DAX, diagnosticar problemas y determinar cuándo es necesario escalar el clúster. La comprobación periódica de las métricas clave le ayuda a mantener el rendimiento, la estabilidad y la rentabilidad al escalar el clúster para que se adapte a los requisitos de su carga de trabajo. Para obtener más información sobre la monitorización de DAX, consulte Monitoreo de la producción.

La siguiente lista incluye algunas de las métricas clave que debe monitorizar:

- Tasa de aciertos de caché: muestra la eficacia con la que DAX sirve los datos en caché, lo que reduce la necesidad de acceder a las tablas subyacentes de DynamoDB. Si hay pocos errores de caché en el clúster, eso indica que hay una buena eficiencia de almacenamiento en caché. Sin embargo, si hay pocos aciertos de caché, eso sugiere que es posible que deba revisitar la configuración de TTL de almacenamiento en caché o que la carga de trabajo no sea adecuada para el almacenamiento en caché.

  Utilice Amazon CloudWatch para calcular la tasa de aciertos de caché del clúster de DAX. Compare las métricas `ItemCacheHits`, `ItemCacheMisses`, `QueryCacheHits` y `QueryCacheMisses` para obtener esta tasa. La siguiente fórmula muestra cómo se calcula la tasa de aciertos de caché. Para calcular la tasa con esta fórmula, divida los aciertos de caché por la suma de los aciertos y errores de caché.

  ```
  Cache hit ratio = Cache hits / (Cache hits + Cache misses)
  ```

  La tasa de aciertos de caché es un número entre 0 y 1, que se representa como un porcentaje. Un porcentaje más alto indica una mejor utilización general de la caché.

- ErrorRequestCount: número de solicitudes que dieron lugar a errores de usuario notificados por el nodo o el clúster. `ErrorRequestCount` incluye las solicitudes sometidas a una limitación por el nodo o el clúster. La monitorización los errores de los usuarios puede ayudarle a identificar errores de escalado, configuraciones erróneas o patrones de elementos o particiones “calientes” en su aplicación.
- Latencias de operaciones: monitorizar la latencia de las operaciones de lectura y escritura hacia el clúster de DAX y desde él puede ayudarle a identificar los cuellos de botella en el rendimiento. El aumento de las latencias puede indicar problemas con la configuración del clúster de DAX, la red o la necesidad de escalarlo.
- Consumo de red: controle las métricas `NetworkBytesIn` y `NetworkBytesOut` para monitorizar el tráfico de red del clúster de DAX. Un aumento inesperado en el rendimiento de la red podría implicar más solicitudes de clientes o patrones de consulta ineficientes que provocarían la transferencia de más datos.

  La monitorización del consumo de la red le ayuda a administrar los costos de su clúster de DAX. También garantiza que la red no se convierta en un cuello de botella para el rendimiento del clúster.
- Tasa de expulsiones: muestra la frecuencia con la que se eliminan elementos de la caché para dejar espacio para otros nuevos. Si la tasa de expulsiones aumenta con el tiempo, es posible que la caché sea demasiado pequeña o que la estrategia de almacenamiento en caché no sea eficaz.

  Monitorice la métrica `EvictedSize` en CloudWatch para determinar si el tamaño de la caché es adecuado para su carga de trabajo. Si el tamaño total expulsado sigue aumentando, es posible que necesite escalar verticalmente el clúster de DAX para alojar una caché más grande.
- Utilización de CPU: se refiere al porcentaje de utilización de CPU del nodo o el clúster. Esta métrica es fundamental para monitorizar cualquier base de datos o sistema de almacenamiento en caché. Un uso elevado de la CPU puede provocar que el clúster de DAX se sobrecargue y necesite escalarse para hacer frente al aumento de la demanda.

  Monitorice la métrica `CPUUtilization` para el clúster de DAX. Si el uso de la CPU se acerca o supera constantemente el 70 o el 80 %, considere la posibilidad de escalar verticalmente el clúster de DAX tal y como se describe en la siguiente sección.

  Si el número de solicitudes que se envían a DAX supera la capacidad de un nodo, DAX limita la velocidad a la que acepta solicitudes adicionales. Para ello, devuelve una ThrottlingException. DAX evalúa continuamente la utilización de la CPU de su clúster para determinar el volumen de solicitudes que puede procesar manteniendo el clúster en buen estado.

Puede monitorizar la métrica `ThrottledRequestCount` que DAX publica en Amazon CloudWatch. Si ve estas excepciones periódicamente, debería plantearse escalar el clúster.

Escalado del clúster de DAX mediante datos de monitorización

Puede determinar si necesita escalar verticalmente o reducir verticalmente el clúster de DAX monitorizando sus métricas de rendimiento.

- Escalado o reducción vertical: si el clúster de DAX tiene un uso elevado de CPU, pocos aciertos de caché (tras optimizar la estrategia de almacenamiento en caché) o altas latencias de las operaciones, debe escalar verticalmente el clúster. Agregar más nodos, lo que también se denomina escalar horizontalmente, puede ayudarle a distribuir la carga de manera más uniforme. Para cargas de trabajo con escrituras por segundo cada vez más altas, es posible que tenga que elegir nodos más potentes (escalado vertical).
- Reducción vertical: si observa que el uso de la CPU y las latencias de operación están por debajo de sus umbrales de manera constante, es posible que haya aprovisionado recursos en exceso. En esos casos, reduzca verticalmente los nodos para reducir los costos. Puede reducir la cantidad de nodos a 1 durante los periodos de baja utilización, pero no puede cerrar el clúster por completo.

Uso de DynamoDB con otros servicios de AWS

Amazon DynamoDB se integra con otros servicios de AWS, lo que permite automatizar las tareas repetitivas o crear aplicaciones que abarcan varios servicios.

Temas

- Configuración de las credenciales de AWS mediante Amazon Cognito para DynamoDB
- Integración con Amazon Redshift
- Procesamiento de los datos de DynamoDB con Apache Hive en Amazon EMR
- Integración de DynamoDB con Amazon S3
- Integración sin ETL de DynamoDB con Amazon SageMaker Lakehouse
- Integración sin ETL de DynamoDB con Amazon OpenSearch Service
- Integración de DynamoDB con Amazon EventBridge
- Integración de DynamoDB con Amazon Managed Streaming para Apache Kafka
- Prácticas recomendadas para la integración con DynamoDB

Configuración de las credenciales de AWS mediante Amazon Cognito para DynamoDB

La manera recomendada de obtener las credenciales de AWS para las aplicaciones web y para móviles es utilizar Amazon Cognito. Amazon Cognito le evita tener que codificar de forma rígida las credenciales de AWS en los archivos. Utiliza los roles de AWS Identity and Access Management (IAM) con el fin de generar credenciales temporales para los usuarios autenticados y no autenticados de la aplicación.

Por ejemplo, si desea configurar los archivos de JavaScript de modo que utilicen un rol sin autenticar de Amazon Cognito para acceder al servicio web de Amazon DynamoDB, haga lo siguiente:

Para configurar credenciales para integrarlas con Amazon Cognito

1. Cree un grupo de identidades de Amazon Cognito que permita identidades no autenticadas.

```
aws cognito-identity create-identity-pool \
    --identity-pool-name DynamoPool \
    --allow-unauthenticated-identities \
```

```
    --output json
{
    "IdentityPoolId": "us-west-2:12345678-1ab2-123a-1234-a12345ab12",
    "AllowUnauthenticatedIdentities": true,
    "IdentityPoolName": "DynamoPool"
}
```

2. Copie la política siguiente en un archivo denominado `myCognitoPolicy.json`. Cambie el identificador del grupo de identidades (*`us-west-2:12345678-1ab2-123a-1234-a12345ab12`*) por su propio `IdentityPoolId` obtenido en el paso anterior:

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "cognito-identity.amazonaws.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "cognito-identity.amazonaws.com:aud": "us-
west-2:12345678-1ab2-123a-1234-a12345ab12"
        },
        "ForAnyValue:StringLike": {
          "cognito-identity.amazonaws.com:amr": "unauthenticated"
        }
      }
    }
  ]
}
```

3. Crear un rol de IAM que asume la política anterior. De este modo, Amazon Cognito se convierte en una entidad de confianza que puede asumir el rol `Cognito_DynamoPoolUnauth`.

```
aws iam create-role --role-name Cognito_DynamoPoolUnauth \
--assume-role-policy-document file://PathToFile/myCognitoPolicy.json --output json
```

4. Conceda al rol `Cognito_DynamoPoolUnauth` acceso pleno al servicio de DynamoDB asociándolo una política administrada (`AmazonDynamoDBFullAccess`).

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonDynamoDBFullAccess \
--role-name Cognito_DynamoPoolUnauth
```

> **Note**
>
> Si lo prefiere, puede conceder acceso preciso a DynamoDB. Para obtener más información, consulte Uso de las condiciones de la política de IAM para el control del acceso preciso.

5. Obtener y copiar el Nombre de recurso de Amazon (ARN) del rol de IAM.

```
aws iam get-role --role-name Cognito_DynamoPoolUnauth --output json
```

6. Agregue el rol `Cognito_DynamoPoolUnauth` al grupo de identidades `DynamoPool`. El formato que debe especificar es `KeyName=string`, donde `KeyName` es `unauthenticated` y string es el ARN del rol obtenido en el paso anterior.

```
aws cognito-identity set-identity-pool-roles \
--identity-pool-id "us-west-2:12345678-1ab2-123a-1234-a12345ab12" \
--roles unauthenticated=arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth --
output json
```

7. Especifique las credenciales de Amazon Cognito en sus archivos. Modifique los valores de `IdentityPoolId` y `RoleArn` en consecuencia.

```
AWS.config.credentials = new AWS.CognitoIdentityCredentials({
IdentityPoolId: "us-west-2:12345678-1ab2-123a-1234-a12345ab12",
RoleArn: "arn:aws:iam::123456789012:role/Cognito_DynamoPoolUnauth"
});
```

Ahora, puede ejecutar sus programas de JavaScript en el servicio web de DynamoDB utilizando las credenciales de Amazon Cognito. Para obtener más información, consulte Configuración de las credenciales en un navegador web en la Guía de inicio de AWS SDK para JavaScript.

Integración con Amazon Redshift

Amazon RedShift es un servicio de almacenamiento de datos de escala de petabyte rápido, totalmente administrado, que hace que sea simple y rentable analizar de manera eficiente todos sus datos utilizando sus herramientas de inteligencia empresariales existentes.

DynamoDB y Amazon Redshift se pueden usar juntos para abordar diferentes necesidades de almacenamiento y procesamiento de datos dentro de una aplicación o un ecosistema de datos.

Consulte los siguientes temas para obtener información más detallada sobre cómo integrar DynamoDB con Amazon Redshift.

Temas

- Consideraciones sobre la integración entre regiones con CMK
- Integración sin ETL de DynamoDB con Amazon Redshift
- Carga de datos desde DynamoDB en Amazon Redshift con el comando COPY

Consideraciones sobre la integración entre regiones con CMK

Cuando intenta realizar la integración de DynamoDB en Amazon Redshift, la acción inicial se lanza desde Amazon Redshift. Sin los permisos adecuados, esta acción podría provocar un error silencioso. En las siguientes secciones se detallan los permisos necesarios para esta integración entre cuentas.

Permisos y políticas de AWS KMS necesarios

Reemplace los marcadores de posición siguientes en los ejemplos:

- `111122223333`: el ID de Cuenta de AWS en el que se aloja Amazon Redshift
- `444455556666`: el ID de Cuenta de AWS en el que se aloja DynamoDB
- `REDSHIFT_ROLE_NAME`: el nombre del rol de IAM que utiliza Amazon Redshift
- `REGION`: la Región de AWS en la que se ubican los recursos
- `TABLE_NAME`: el nombre de la tabla de DynamoDB
- `KMS_KEY_ID`: el ID de la clave de KMS

Política de claves de KMS en la cuenta de DynamoDB

La siguiente política de claves de AWS KMS permite el acceso entre cuentas entre los servicios de DynamoDB y Amazon Redshift. En este ejemplo, la cuenta 444455556666 contiene la tabla y la clave de AWS KMS de DynamoDB, mientras que la cuenta 111122223333 contiene el clúster de Amazon Redshift al que se debe acceder para descifrar los datos.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow Redshift to use the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/REDSHIFT_ROLE_NAME"
            },
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": "*"
        }
    ]
}
```

Política de IAM para el rol de Amazon Redshift (en la cuenta de Amazon Redshift)

La siguiente política de IAM permite que un servicio de Amazon Redshift acceda a las tablas de DynamoDB y a sus claves de cifrado de AWS KMS asociadas en un escenario con varias cuentas. En este ejemplo, la cuenta 444455556666 contiene las claves y los recursos de AWS KMS de DynamoDB a los que necesita acceder el servicio de Amazon Redshift.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDynamoDBAccess",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:BatchGetItem",
                "dynamodb:GetItem",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:444455556666:table/TABLE_NAME",
                "arn:aws:dynamodb:*:444455556666:table/TABLE_NAME/stream/*"
            ]
        },
        {
            "Sid": "AllowKMSAccess",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": "arn:aws:kms:us-east-1:444455556666:key/KMS_KEY_ID"
```

```
        }
    ]
}
```

Relación de confianza para el rol de Amazon Redshift

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Política de tabla de DynamoDB (si se utilizan políticas basadas en recursos)

La siguiente política basada en recursos permite que un servicio de Amazon Redshift de la cuenta 111122223333 acceda a las tablas y transmisiones de DynamoDB en la cuenta 444455556666. Adjunte esta política a la tabla de DynamoDB para permitir el acceso entre cuentas.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRedshiftAccess",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/REDSHIFT_ROLE_NAME"
            },
```

```
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:BatchGetItem",
                "dynamodb:Scan",
                "dynamodb:Query",
                "dynamodb:BatchGetItem",
                "dynamodb:GetItem",
                "dynamodb:GetRecords",
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams"
            ],
            "Resource": [
                "arn:aws:dynamodb:*:444455556666:table/TABLE_NAME",
                "arn:aws:dynamodb:*:444455556666:table/TABLE_NAME/stream/*"
            ]
        }
    ]
}
```

Consideraciones importantes

1. Asegúrese de que la clave de KMS esté en la misma región que la tabla de DynamoDB.
2. La clave de KMS debe ser una clave administrada por el cliente (CMK), no una Clave administrada de AWS.
3. Si está utilizando tablas globales de DynamoDB, configure los permisos para todas las regiones pertinentes.
4. Considere agregar declaraciones de condición para restringir el acceso en función de los puntos de conexión de VPC o los rangos de IP.
5. Para mejorar la seguridad, considere el uso de la condición `aws:PrincipalOrgID` para restringir el acceso a la organización.
6. Supervise el uso de la clave de KMS mediante métricas de CloudTrail y CloudWatch.

Integración sin ETL de DynamoDB con Amazon Redshift

La integración sin ETL de Amazon DynamoDB con Amazon Redshift habilita un análisis sin problemas de los datos de DynamoDB sin necesidad de codificación. Esta característica totalmente administrada replica automáticamente las tablas de DynamoDB en una base de datos de Amazon

Redshift para que los usuarios puedan ejecutar consultas y análisis SQL en sus datos de DynamoDB sin tener que configurar procesos ETL complejos. La integración funciona mediante la replicación de los datos de la tabla de DynamoDB en la base de datos de Amazon Redshift.

Para configurar la integración, simplemente especifique una tabla de DynamoDB como origen y una base de datos de Amazon Redshift como destino. Tras la activación, la integración exporta la tabla completa de DynamoDB para rellenar la base de datos de Amazon Redshift. El tiempo que tarda este proceso inicial en completarse depende del tamaño de tabla de DynamoDB. A continuación, la integración sin ETL replica de forma incremental las actualizaciones de DynamoDB a Amazon Redshift cada 15-30 minutos mediante las exportaciones incrementales de DynamoDB. Esto significa que los datos de DynamoDB replicados en Amazon Redshift se mantienen actualizados automáticamente.

Una vez configurados, los usuarios pueden analizar los datos de DynamoDB en Amazon Redshift mediante herramientas y clientes SQL estándar, sin que ello afecte al rendimiento de las tablas de DynamoDB. Al eliminar el engorroso ETL, esta integración sin ETL proporciona una forma rápida y sencilla de obtener información de DynamoDB a través de las capacidades de análisis y de machine learning de Amazon Redshift.

Temas

- Requisitos previos a la creación de una integración sin ETL de DynamoDB con Amazon Redshift
- Limitaciones al utilizar las integraciones sin ETL de DynamoDB con Amazon Redshift
- Creación de una integración sin ETL de Amazon DynamoDB con Amazon Redshift
- Visualización de las integraciones sin ETL de DynamoDB con Amazon Redshift
- Eliminación de integraciones sin ETL de DynamoDB con Amazon Redshift

Requisitos previos a la creación de una integración sin ETL de DynamoDB con Amazon Redshift

1. Debe haber creado la tabla de DynamoDB de origen y el clúster de Amazon Redshift de destino antes de crear una integración. Esta información se explica con más detalle en Paso 1: Configurar una tabla de DynamoDB de origen y Paso 2: Creación de un almacenamiento de datos de Amazon Redshift.
2. Una integración sin ETL entre Amazon DynamoDB y Amazon Redshift requiere que la tabla de DynamoDB de origen tenga habilitada la recuperación en un momento dado (PITR).

3. En el caso de las políticas basadas en recursos, si crea la integración en la que la tabla de DynamoDB y el almacenamiento de datos de Amazon Redshift estén en la misma cuenta, puede utilizar la opción Soluciónalo por mí durante el paso de creación de la integración para aplicar automáticamente las políticas de recursos necesarias tanto a DynamoDB como a Amazon Redshift.

   Si crea una integración en la que la tabla de DynamoDB y el almacenamiento de datos de Amazon Redshift estén en cuentas de AWS diferentes, tendrá que aplicar la siguiente política de recursos en la tabla de DynamoDB.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid":
 "StatementthatallowsAmazonRedshiftservicetoDescribeTableandExportTable",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
            },
            "Action": [
                "dynamodb:ExportTableToPointInTime",
                "dynamodb:DescribeTable"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
                }
            }
        },
        {
            "Sid":
 "StatementthatallowsAmazonRedshiftservicetoDescribeTableandExportTable",
            "Effect": "Allow",
            "Principal": {
                "Service": "redshift.amazonaws.com"
```

```
            },
            "Action": "dynamodb:DescribeExport",
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/table-
name/export/*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
                }
            }
        }
    ]
}
```

También es posible que deba configurar la política de recursos en el almacenamiento de datos de Amazon Redshift. Para obtener más información, consulte Configuración de la autorización mediante la API de Amazon Redshift.

4. Para las políticas basadas en identidades:

   a. El usuario que cree la integración necesita una política basada en identidades que autorice las siguientes acciones: `GetResourcePolicy`, `PutResourcePolicy` y `UpdateContinuousBackups`.

> **Note**
>
> En los siguientes ejemplos de políticas, el recurso se mostrará como `arn:aws:redshift{-serverless}`. Este es un ejemplo para demostrar que el ARN puede ser `arn:aws:redshift` o `arn:aws:redshift-serverless`, dependiendo de si el espacio de nombres es un clúster de Amazon Redshift o un espacio de nombres de Amazon Redshift sin servidor.

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:ListTables"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetResourcePolicy",
                "dynamodb:PutResourcePolicy",
                "dynamodb:UpdateContinuousBackups"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-east-1:111122223333:table/table-name"
            ]
        },
        {
            "Sid": "AllowRedshiftDescribeIntegration",
            "Effect": "Allow",
            "Action": [
                "redshift:DescribeIntegrations"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowRedshiftCreateIntegration",
            "Effect": "Allow",
            "Action": "redshift:CreateIntegration",
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:integration:*"
        },
        {
            "Sid": "AllowRedshiftModifyDeleteIntegration",
            "Effect": "Allow",
            "Action": [
                "redshift:ModifyIntegration",
                "redshift:DeleteIntegration"
            ],
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:integration:uuid"
        },
```

```
        {
            "Sid": "AllowRedshiftCreateInboundIntegration",
            "Effect": "Allow",
            "Action": "redshift:CreateInboundIntegration",
            "Resource": "arn:aws:redshift:us-
east-1:111122223333:namespace:uuid"
        }
    ]
}
```

b. El usuario responsable de configurar el espacio de nombres de Amazon Redshift de destino necesita una política basada en identidades que autorice las siguientes acciones: `PutResourcePolicy`, `DeleteResourcePolicy` y `GetResourcePolicy`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "redshift:PutResourcePolicy",
                "redshift:DeleteResourcePolicy",
                "redshift:GetResourcePolicy"
            ],
            "Resource": [
                "arn:aws:redshift:us-east-1:111122223333:cluster:cluster-
name"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "redshift:DescribeInboundIntegrations"
            ],
            "Resource": [
                "arn:aws:redshift:us-east-1:111122223333:cluster:cluster-
name"
            ]
        }
    ]
}
```

5. Permisos de la clave de cifrado

   Si la tabla de DynamoDB de origen está cifrada con una clave de AWS KMS administrada por el cliente, tendrá que agregar la siguiente política a la clave de KMS. Esta política permite a Amazon Redshift exportar datos de la tabla cifrada con la clave de KMS.

```
{
    "Sid": "Statement to allow Amazon Redshift service to perform Decrypt operation
 on the source DynamoDB Table",
    "Effect": "Allow",
    "Principal": {
        "Service": [
            "redshift.amazonaws.com"
        ]
    },
    "Action": "kms:Decrypt",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:SourceAccount": "<account>"
        },
        "ArnEquals": {
            "aws:SourceArn": "arn:aws:redshift:<region>:<account>:integration:*"
        }
    }
}
```

También puede seguir los pasos de Introducción a las integraciones sin ETL de la Guía de administración de Amazon Redshift para configurar los permisos del espacio de nombres de Amazon Redshift.

Limitaciones al utilizar las integraciones sin ETL de DynamoDB con Amazon Redshift

La versión actual de esta integración presenta las siguientes limitaciones generales. Estas limitaciones pueden cambiar en versiones posteriores.

> **Note**
>
> Además de las limitaciones que se indican a continuación, revise también las consideraciones generales al utilizar integraciones sin ETL en Consideraciones al utilizar

las integraciones sin ETL con Amazon Redshift en la Guía de administración de Amazon Redshift.

- La tabla de DynamoDB y el clúster de Amazon Redshift deben estar en la misma región.
- La tabla de DynamoDB de origen debe estar cifrada con una clave de AWS KMS administrada por el cliente o propiedad de Amazon. El cifrado administrado por Amazon no es compatible con la tabla de DynamoDB de origen.

Creación de una integración sin ETL de Amazon DynamoDB con Amazon Redshift

Antes de crear una integración sin ETL, primero debe configurar la tabla de DynamoDB de origen y, a continuación, el almacenamiento de datos de Amazon Redshift de destino.

Paso 1: Configurar una tabla de DynamoDB de origen

Para crear una integración sin ETL con Amazon Redshift, debe habilitar la recuperación en un momento dado (PITR) en su tabla. Si no tiene el PITR activado, la consola puede solucionarlo durante el proceso de configuración de la integración. Para más detalles sobre cómo activar PITR, consulte Recuperación en un momento dado.

Paso 2: Creación de un almacenamiento de datos de Amazon Redshift

Si aún no tiene un almacenamiento de datos de Amazon Redshift, puede crear uno. Para crear un grupo de trabajo de Amazon Redshift sin servidor, consulte Creación de un grupo de trabajo con un espacio de nombres. Para crear un clúster de Amazon Redshift, consulte Creación de un clúster.

Para que la integración funcione, el clúster o el grupo de trabajo de Amazon Redshift de destino debe tener el parámetro enable_case_sensitive_identifier activado. Para obtener más información sobre cómo habilitar la distinción entre mayúsculas y minúsculas, consulte Turn on case sensitivity for your data warehouse en la Guía de administración de Amazon Redshift.

Una vez completada la configuración del grupo de trabajo o clúster de Amazon Redshift, debe configurar el almacenamiento de datos. Para obtener más información, consulte Integraciones sin ETL en la Guía de administración de Amazon Redshift.

Paso 3: Creación de una integración sin ETL de DynamoDB

Antes de crear una integración sin ETL, asegúrese de completar las tareas de la sección titulada Requisitos previos a la creación de una integración sin ETL de DynamoDB con Amazon Redshift. El

proceso de crear una integración entre DynamoDB y Amazon Redshift consta de dos pasos. Primero cree una integración desde DynamoDB y, a continuación, asocie una base de datos de Amazon Redshift a esta integración recién creada.

Creación de una integración sin ETL

1. Inicie sesión en la Consola de administración de AWS y abra la consola de Amazon DynamoDB en https://console.aws.amazon.com/dynamodbv2.
2. En el panel de navegación, elija Integraciones.
3. Seleccione Crear integración sin ETL y elija Amazon Redshift.
4. Esto le redirige a la consola de Amazon Redshift. Para continuar con el procedimiento, consulte la sección DynamoDB en Crear una integración sin ETL para DynamoDB.

Visualización de las integraciones sin ETL de DynamoDB con Amazon Redshift

Puede ver los detalles de una integración sin ETL para ver su información de configuración y su estado actual.

Para ver los detalles de una integración sin ETL en la consola de Amazon DynamoDB:

1. Inicie sesión en la Consola de administración de AWS y abra la consola de Amazon DynamoDB en https://console.aws.amazon.com/dynamodbv2.
2. En la consola de DynamoDB, elija Integraciones.
3. En el panel Integración sin ETL, seleccione la integración sin ETL que desee ver.

Para ver los detalles de una integración sin ETL en la consola de Amazon Redshift:

1. Inicie sesión en la Consola de administración de AWS y abra la consola de Amazon Redshift en https://console.aws.amazon.com/redshiftv2.
2. Siga los pasos que se indican en Visualización de integraciones sin ETL.

> **Note**
>
> Los posibles estados de una integración sin ETL con Amazon Redshift se indican en Visualización de integraciones sin ETL de la Guía de administración de Amazon Redshift.

Eliminación de integraciones sin ETL de DynamoDB con Amazon Redshift

Al eliminar una integración sin ETL, los datos no se eliminan de DynamoDB o Amazon Redshift, sino que DynamoDB deja de enviar datos de la tabla de origen al destino de Amazon Redshift.

Eliminación de una integración sin ETL

1. Inicie sesión en la Consola de administración de AWS y abra la consola de Amazon DynamoDB en https://console.aws.amazon.com/dynamodbv2.
2. En la consola de DynamoDB, elija Integraciones.
3. En el panel Integración sin ETL, seleccione la integración sin ETL que desee eliminar.
4. Elija Administrar. El enlace le dirigirá a la página de detalles de la integración.
5. Para confirmar la eliminación, elija Eliminar.

Carga de datos desde DynamoDB en Amazon Redshift con el comando COPY

Amazon Redshift trabaja con Amazon DynamoDB con prestaciones avanzadas de inteligencia empresarial y una potente interfaz basada en SQL. Cuando se copian datos de una tabla de DynamoDB en Amazon Redshift, se pueden llevar a cabo consultas complejas de análisis de esos datos, lo que incluye uniones con otras tablas del clúster de Amazon Redshift.

En lo que respecta al rendimiento aprovisionado, una operación de copia desde una tabla de DynamoDB consume capacidad de lectura de esa tabla. Después de copiar los datos, las consultas SQL que se llevan a cabo en Amazon Redshift no afectan a DynamoDB de ningún modo. El motivo es que las consultas se ejecutan en una copia de los datos de DynamoDB y no en DynamoDB propiamente dicho.

Para poder cargar datos desde una tabla de DynamoDB, antes debe crear una tabla de Amazon Redshift que actuará como destino de los datos. Tenga en cuenta que, al hacerlo, estará copiando datos de un entorno NoSQL en un entorno SQL y que hay reglas que no se aplican por igual en ambos. A continuación se muestran algunas de las diferencias que deben tenerse en cuenta:

- Los nombres de las tablas de DynamoDB pueden contener un máximo de 255 caracteres, incluidos “.” el punto “_”. Además, distinguen entre mayúsculas y minúsculas. Los nombres de las tablas de Amazon Redshift están limitados a 127 caracteres, no pueden contener puntos ni

guiones y no distinguen entre mayúsculas y minúsculas. Los nombres de las tablas tampoco pueden entrar en conflicto con ninguna palabra reservada de Amazon Redshift.

- DynamoDB no admite el concepto NULL de SQL. Debe especificar cómo debe interpretar Amazon Redshift los valores de atributos vacíos o en blanco de DynamoDB, para que los trate como NULL o como campos vacíos.
- Los tipos de datos de DynamoDB no se corresponden directamente con los de Amazon Redshift. Debe asegurarse de que cada columna de la tabla de Amazon Redshift sea del tipo y el tamaño de datos correctos para adaptarse a los datos de DynamoDB.

A continuación se muestra un ejemplo de comando COPY de SQL en Amazon Redshift:

```
copy favoritemovies from 'dynamodb://my-favorite-movies-table'
credentials 'aws_access_key_id=<Your-Access-Key-ID>;aws_secret_access_key=<Your-Secret-Access-Key>'
readratio 50;
```

En este ejemplo, la tabla de origen de DynamoDB es `my-favorite-movies-table`. La tabla de destino de Amazon Redshift es`favoritemovies`. La cláusula `readratio 50` regula el porcentaje de desempeño provisionado que se consume; en este caso, el comando COPY utilizará como máximo un 50 % de las unidades de capacidad de lectura provisionadas para `my-favorite-movies-table`. Recomendamos encarecidamente establecer este porcentaje en un valor menor que el promedio de desempeño provisionado sin utilizar.

Para obtener instrucciones detalladas sobre cómo cargar datos desde DynamoDB en Amazon Redshift, consulte las secciones siguientes de la Guía para desarrolladores de bases de datos Amazon Redshift:

- Cargar datos desde una tabla de DynamoDB
- The COPY command
- Ejemplos de COPY

Procesamiento de los datos de DynamoDB con Apache Hive en Amazon EMR

Amazon DynamoDB se integra con Apache Hive, una aplicación de almacén de datos que se ejecuta en Amazon EMR. Hive puede leer y escribir datos en las tablas de DynamoDB, lo que le permite:

- Consultar datos de DynamoDB en directo utilizando un lenguaje semejante a SQL (HiveQL).
- Copiar datos de una tabla de DynamoDB en un bucket de Amazon S3 y viceversa.
- Copiar datos de una tabla de DynamoDB en Hadoop Distributed File System (HDFS) y viceversa.
- Realizar operaciones de unión con las tablas de DynamoDB.

Temas

- Información general
- Tutorial: Uso de Amazon DynamoDB y Apache Hive
- Creación de una tabla externa en Hive
- Procesamiento de instrucciones de HiveQL
- Consulta de datos en DynamoDB
- Copia de datos en y desde Amazon DynamoDB
- Ajuste del rendimiento

Información general

Amazon EMR es un servicio que facilita el procesamiento rápido y rentable de enormes volúmenes de datos. Para utilizar Amazon EMR, se lanza un clúster administrado de instancias de Amazon EC2 que ejecutan el marco de trabajo de código abierto Hadoop. Hadoop es una aplicación distribuida que implementa el algoritmo de MapReduce, en virtud del cual se mapea una tarea a varios nodos del clúster. En paralelo con los demás nodos, cada nodo procesa el trabajo que se ha designado para él. Por último, las salidas se reducen a un solo nodo, que produce el resultado final.

Puede lanzar el clúster de Amazon EMR de modo que sea persistente o transitorio:

- Un clúster persistente se ejecuta hasta que se cierra. Los clústeres persistentes son idóneos para el análisis de datos, el almacenamiento de datos o cualquier otro uso interactivo.
- Un clúster transitorio se ejecuta el tiempo suficiente para procesar un flujo de trabajo y, a continuación, se cierra automáticamente. Los clústeres transitorios son idóneos para las tareas de procesamiento periódicas, tales como la ejecución de scripts.

Para obtener más información sobre la arquitectura y administración de Amazon EMR, consulte la Guía de administración de Amazon EMR.

Cuando se lanza un clúster de Amazon EMR, se especifica el número inicial y el tipo de instancias de Amazon EC2. También se especifican otras aplicaciones distribuidas (además de Hadoop) que se desea ejecutar en el clúster. Estas aplicaciones son, entre otras, Hue, Mahout, Pig o Spark.

Para obtener información sobre las aplicaciones para Amazon EMR, consulte la Guía de emisión de Amazon EMR.

Según la configuración del clúster, puede que disponga de uno o varios de los tipos de nodos siguientes:

- Nodo líder: administra el clúster coordinando la distribución del ejecutable de MapReduce y de los subconjuntos de datos sin procesar, hasta los grupos de instancias principal y de tareas. También hace un seguimiento del estado de cada tarea realizada y monitorea la salud de los grupos de instancias. Solo hay un nodo líder en un clúster.
- Nodos principales: ejecutan tareas de MapReduce y almacenan datos utilizando el Hadoop Distributed File System (HDFS).
- Nodos de tareas (opcionales): ejecutan las tareas de MapReduce.

Tutorial: Uso de Amazon DynamoDB y Apache Hive

En este tutorial, lanzaremos un clúster de Amazon EMR y, a continuación, usaremos Apache Hive para procesar los datos almacenados en una tabla de DynamoDB.

Hive es una aplicación de almacenamiento de datos para Hadoop que permite procesar y analizar datos de varios orígenes. Hive proporciona un lenguaje similar a SQL, HiveQL, que permite trabajar con datos almacenados localmente en el clúster de Amazon EMR o en un origen de datos externo (como Amazon DynamoDB).

Para obtener más información, consulte el Hive Tutorial.

Temas

- Antes de empezar
- Paso 1: Crear un par de claves de Amazon EC2
- Paso 2: lanzar un clúster de Amazon EMR
- Paso 3: conectarse al nodo principal
- Paso 4: cargar los datos en HDFS
- Paso 5: copiar los datos a DynamoDB

- Paso 6: consultar los datos en la tabla de DynamoDB
- Paso 7: limpieza (opcional)

Antes de empezar

Para este tutorial, necesitará lo siguiente:

- Una cuenta de AWS. Si no dispone de una, consulte Inscripción en AWS.
- Un cliente SSH (Secure Shell). El cliente SSH se utiliza para conectarse al nodo líder del clúster de Amazon EMR y ejecutar comandos interactivos. Los clientes SSH están disponibles de forma predeterminada en la mayoría de las instalaciones de Linux, Unix y Mac OS X. Los usuarios de Windows pueden descartar e instalar el cliente PuTTY, que es compatible con SSH.

Siguiente paso

Paso 1: Crear un par de claves de Amazon EC2

Paso 1: Crear un par de claves de Amazon EC2

En este paso, crearemos el par de claves de Amazon EC2 que se requiere para conectarse a un nodo líder de Amazon EMR y ejecutar comandos de Hive.

1. Inicie sesión en la Consola de administración de AWS y abra la consola de Amazon EC2 en https://console.aws.amazon.com/ec2/.
2. Elija una región (por ejemplo, `US West (Oregon)`). Debe ser la misma región en la que se encuentra la tabla de DynamoDB.
3. En el panel de navegación, seleccione Key Pairs (Pares de claves).
4. Seleccione Create Key Pair.
5. En Key pair name, escriba un nombre para el par de claves (por ejemplo, `mykeypair`) y después elija Create.
6. Descargue el archivo de clave privada. El nombre de archivo terminará con `.pem` (por ejemplo, `mykeypair.pem`). Mantenga este archivo de clave privada en un lugar seguro. Lo necesitará para acceder a cualquier clúster de Amazon EMR que lance con este par de claves.

> **Important**
>
> Si pierde el par de clave, no podrá conectarse al nodo líder del clúster de Amazon EMR.

Para obtener más información sobre los pares de claves, consulte Pares de claves de Amazon EC2 en la Guía del usuario de Amazon EC2.

Siguiente paso

Paso 2: lanzar un clúster de Amazon EMR

Paso 2: lanzar un clúster de Amazon EMR

En este paso, configuraremos y lanzaremos un clúster de Amazon EMR. Ya estarán instalados en el clúster tanto Hive como un controlador de almacenamiento para DynamoDB.

1. Abra la consola de Amazon EMR enhttps://console.aws.amazon.com/emr.
2. Elija Create Cluster (Crear clúster).
3. En la página Create Cluster - Quick Options, haga lo siguiente:
   a. En Cluster name, escriba el nombre del clúster (por ejemplo, `My EMR cluster`).
   b. En EC2 key pair, elija el par de claves que creó anteriormente.

   No cambie los valores predeterminados de los demás ajustes.
4. Elija Create cluster.

Se tardan unos minutos en lanzar el clúster. Puede utilizar la página Cluster Details (Detalles del clúster) de la consola de Amazon EMR para monitorear el progreso.

Cuando el estado cambia a `Waiting`, el clúster está preparado para usarlo.

Archivos del registro del clúster y Amazon S3

Un clúster de Amazon EMR genera archivos de registros que contienen información acerca del estado del clúster y sobre depuración. La configuración predeterminada de Create Cluster - Quick Options (Crear un clúster: opciones rápidas) incluye la configuración de registros de Amazon EMR.

Si no hay un bucket de Amazon S3, la Consola de administración de AWS lo crea. El nombre del bucket es `aws-logs-`*`account-id`*`-`*`region`*, donde *`account-id`* es el número de su cuenta de AWS y *`region`* es la región donde lanzó el clúster (por ejemplo, `aws-logs-123456789012-us-west-2`).

> **Note**
>
> Puede usar la consola de Amazon S3 para ver los archivos de registro. Para obtener más información, consulte View Log Files (Ver archivos de registros) en la Guía de administración de Amazon EMR.

Puede usar este bucket para otros fines, además de generar registros. Por ejemplo, puede utilizar el bucket como ubicación para almacenar un script de Hive o como destino al exportar datos de Amazon DynamoDB a Amazon S3.

Siguiente paso

Paso 3: conectarse al nodo principal

Paso 3: conectarse al nodo principal

Cuando el estado del clúster de Amazon EMR cambia a `Waiting`, ya puede conectarse al nodo líder mediante SSH y llevar a cabo operaciones de línea de comandos.

1. En la consola de Amazon EMR, elija el nombre del clúster para ver su estado.
2. En la página Cluster Details (Detalles del clúster), busque el campo Leader public DNS (DNS público del líder). Se trata del nombre de DNS público del nodo líder del clúster de Amazon EMR.
3. A la derecha del nombre de DNS, elija el enlace SSH.
4. Siga las instrucciones de Conectarse al nodo líder mediante SSH.

   Según cuál sea su sistema operativo, elija la pestaña Windows o Mac/Linux. A continuación, siga las instrucciones para conectarse al nodo líder.

Después de conectarse al nodo líder mediante SSH o PuTTY, debe aparecer un símbolo del sistema parecido al siguiente:

```
[hadoop@ip-192-0-2-0 ~]$
```

Siguiente paso

Paso 4: cargar los datos en HDFS

Paso 4: cargar los datos en HDFS

En este paso, copiaremos un archivo de datos en Hadoop Distributed File System (HDFS) y, a continuación, crearemos una tabla Hive externa mapeada con ese archivo de datos.

Descarga del ejemplo de datos

1. Descargue el ejemplo de archivo de datos (`features.zip`):

   ```
   wget https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
   features.zip
   ```

2. Extraiga el archivo `features.txt` del archivo comprimido:

   ```
   unzip features.zip
   ```

3. Vea las primeras líneas del archivo `features.txt`:

   ```
   head features.txt
   ```

   El resultado debe ser parecido al siguiente:

   ```
   1535908|Big Run|Stream|WV|38.6370428|-80.8595469|794
   875609|Constable Hook|Cape|NJ|40.657881|-74.0990309|7
   1217998|Gooseberry Island|Island|RI|41.4534361|-71.3253284|10
   26603|Boone Moore Spring|Spring|AZ|34.0895692|-111.410065|3681
   1506738|Missouri Flat|Flat|WA|46.7634987|-117.0346113|2605
   1181348|Minnow Run|Stream|PA|40.0820178|-79.3800349|1558
   1288759|Hunting Creek|Stream|TN|36.343969|-83.8029682|1024
   533060|Big Charles Bayou|Bay|LA|29.6046517|-91.9828654|0
   829689|Greenwood Creek|Stream|NE|41.596086|-103.0499296|3671
   541692|Button Willow Island|Island|LA|31.9579389|-93.0648847|98
   ```

   El archivo `features.txt` contiene un subconjunto de los datos del United States Board on Geographic Names (Consejo Estadounidense de Nombres Geográficos) (http://

geonames.usgs.gov/domestic/download_data.htm). Los campos de cada línea representan lo siguiente:

- Identificador del accidente geográfico (identificador único)
- Nombre
- Clase (lago, bosque, arroyo, etc.)
- Estado
- Latitud (grados)
- Longitud (grados)
- Altitud (en pies)

4. En el símbolo del sistema, escriba el siguiente comando:

```
hive
```

   El símbolo del sistema cambia a lo siguiente: `hive>`

5. Escriba la siguiente instrucción de HiveQL para crear una tabla de Hive nativa:

```
CREATE TABLE hive_features
    (feature_id             BIGINT,
    feature_name            STRING ,
    feature_class           STRING ,
    state_alpha             STRING,
    prim_lat_dec            DOUBLE ,
    prim_long_dec           DOUBLE ,
    elev_in_ft              BIGINT)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '|'
    LINES TERMINATED BY '\n';
```

6. Escriba la siguiente instrucción de HiveQL para cargar datos en la tabla:

```
LOAD DATA
LOCAL
INPATH './features.txt'
OVERWRITE
INTO TABLE hive_features;
```

7. Ahora, tenemos una tabla de Hive nativa que contiene los datos del archivo `features.txt`. Para comprobarlo, escriba la siguiente instrucción de HiveQL:

```
SELECT state_alpha, COUNT(*)
FROM hive_features
GROUP BY state_alpha;
```

El resultado debería ser una lista de estados y el número de accidentes geográficos de cada uno de ellos.

Siguiente paso

[Paso 5: copiar los datos a DynamoDB]

Paso 5: copiar los datos a DynamoDB

En este paso, copiaremos los datos de la tabla de Hive (`hive_features`) en una nueva tabla de DynamoDB.

1. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. Seleccione Create Table (Crear tabla).
3. En la página Create DynamoDB table, haga lo siguiente:

   a. En la Table (Tabla) escriba **`Features`**.

   b. En Primary key (Clave principal), en el campo Partition key (Clave de partición), escriba **`Id`**. Establezca el tipo de datos en Number (Número).

      Desactive Use Default Settings (Usar configuración predeterminada). En Provisioned Capacity, especifique lo siguiente:

      - Unidades de capacidad de lectura—`10`
      - Unidades de capacidad de escritura—`10`

   Seleccione Crear.
4. En el símbolo del sistema de Hive, escriba la instrucción de HiveQL siguiente:

```
CREATE EXTERNAL TABLE ddb_features
    (feature_id   BIGINT,
    feature_name  STRING,
    feature_class STRING,
```

```
    state_alpha   STRING,
    prim_lat_dec  DOUBLE,
    prim_long_dec DOUBLE,
    elev_in_ft    BIGINT)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES(
    "dynamodb.table.name" = "Features",

 "dynamodb.column.mapping"="feature_id:Id,feature_name:Name,feature_class:Class,state_alpha
);
```

Ahora, hemos establecido un mapeo entre Hive y la tabla Features de DynamoDB.

5. Ingrese la siguiente instrucción de HiveQL para importar los datos a DynamoDB:

```
INSERT OVERWRITE TABLE ddb_features
SELECT
    feature_id,
    feature_name,
    feature_class,
    state_alpha,
    prim_lat_dec,
    prim_long_dec,
    elev_in_ft
FROM hive_features;
```

Hive enviará un trabajo de MapReduce, que se procesará en el clúster de Amazon EMR. El trabajo puede tardará varios minutos en completarse.

6. Compruebe que los datos se han cargado en DynamoDB:

   a. En el panel de navegación de la consola de DynamoDB, elija Tables (Tablas).

   b. Elija la tabla Features y, a continuación, elija la pestaña Items para ver los datos.

Siguiente paso

Paso 6: consultar los datos en la tabla de DynamoDB

Paso 6: consultar los datos en la tabla de DynamoDB

En este paso, vamos a usar HiveQL para consultar la tabla Features en DynamoDB. Pruebe las siguientes consultas en Hive:

1. Todos los tipos de accidentes geográficos (`feature_class`) por orden alfabético:

```
SELECT DISTINCT feature_class
FROM ddb_features
ORDER BY feature_class;
```

2. Todos los lagos que empiezan por la letra "M":

```
SELECT feature_name, state_alpha
FROM ddb_features
WHERE feature_class = 'Lake'
AND feature_name LIKE 'M%'
ORDER BY feature_name;
```

3. Los estados que tienen al menos tres accidentes geográficos con más de una milla de altitud (5280 pies/1609,34 metros):

```
SELECT state_alpha, feature_class, COUNT(*)
FROM ddb_features
WHERE elev_in_ft > 5280
GROUP by state_alpha, feature_class
HAVING COUNT(*) >= 3
ORDER BY state_alpha, feature_class;
```

Siguiente paso

Paso 7: limpieza (opcional)

Paso 7: limpieza (opcional)

Una vez completado el tutorial, puede continuar leyendo esta sección para obtener más información sobre cómo usar los datos de DynamoDB en Amazon EMR. Mientras lo hace, puede que prefiera mantener el clúster de Amazon EMR en funcionamiento.

Sin embargo, cuando ya no necesite el clúster, conviene terminarlo y eliminar todos los recursos asociados. De este modo, evitará que se le cobre por los recursos que no necesita.

1. Termine el clúster de Amazon EMR:

   a. Abra la consola de Amazon EMR enhttps://console.aws.amazon.com/emr.

b. Elija el clúster de Amazon EMR, elija Terminate (Terminar) y, a continuación, confirme la operación.

2. Elimine la tabla Features de DynamoDB:

   a. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
   b. En el panel de navegación, elija Tablas.
   c. Elija la tabla Features. En el menú Actions, elija Delete Table.

3. Elimine el bucket de Amazon S3 que contiene los archivos de registros de Amazon EMR:

   a. Abra la consola de Amazon S3 en https://console.aws.amazon.com/s3/.
   b. En la lista de buckets, elija `aws-logs-` *`accountID`*`-`*`region`*, donde *`accountID`* es el número de su cuenta de AWS y *`region`* es la región en la que ha lanzado el clúster.
   c. En el menú Action, elija Delete.

Creación de una tabla externa en Hive

En Tutorial: Uso de Amazon DynamoDB y Apache Hive, hemos creado una tabla de Hive externa que mapea a una tabla de DynamoDB. Cada vez que emitía instrucciones de HiveQL para la tabla externa, las operaciones de lectura y escritura se transmitían a la tabla de DynamoDB.

Podemos considerar que una tabla externa es un puntero que señala a un origen de datos administrado y almacenado en otro lugar. En este caso, el origen de datos subyacente es una tabla de DynamoDB. (La tabla debe existir previamente. No se puede crear, actualizar ni eliminar una tabla de DynamoDB desde Hive). Utilice la instrucción `CREATE EXTERNAL TABLE` para crear la tabla externa. A partir de ese momento, podrá usar HiveQL para trabajar con los datos de DynamoDB, como si se encontrasen almacenado localmente en Hive.

> **Note**
>
> Puede usar instrucciones `INSERT` para insertar datos en una tabla externa e instrucciones `SELECT` para seleccionar datos en ella. Sin embargo, no se pueden usar instrucciones `UPDATE` ni `DELETE` para manipular los datos de la tabla.

Cuando ya no necesite la tabla externa, puede eliminarla mediante la instrucción `DROP TABLE`. En este caso, `DROP TABLE` solamente elimina la tabla externa en Hive. La operación no afecta a la tabla de DynamoDB subyacente ni a ninguno de los datos que contiene.

Temas

- Sintaxis de CREATE EXTERNAL TABLE
- Mapeos de tipos de datos

Sintaxis de CREATE EXTERNAL TABLE

A continuación se muestra la sintaxis de HiveQL para crear una tabla de Hive externa que mapea a una tabla de DynamoDB:

```
CREATE EXTERNAL TABLE hive_table

 (hive_column1_name hive_column1_datatype, hive_column2_name hive_column2_datatype...)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES (
    "dynamodb.table.name" = "dynamodb_table",
    "dynamodb.column.mapping" =
 "hive_column1_name:dynamodb_attribute1_name,hive_column2_name:dynamodb_attribute2_name..."
);
```

La línea 1 es el principio de la instrucción `CREATE EXTERNAL TABLE`, en la que se indica el nombre de la tabla de Hive (hive_table) que se desea a crear.

La línea 2 especifica las columnas y los tipos de datos de hive_table. Debe definir las columnas y los tipos de datos que se correspondan con los atributos de la tabla de DynamoDB.

La línea 3 es la cláusula `STORED BY`, en la que se especifica una clase que controla la administración de los datos entre Hive y la tabla de DynamoDB. Para DynamoDB, `STORED BY` debe definirse en `'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'`.

La línea 4 es el principio de la cláusula `TBLPROPERTIES`, en la que se definen los parámetros siguientes de `DynamoDBStorageHandler`:

- `dynamodb.table.name`: el nombre de la tabla de DynamoDB.
- `dynamodb.column.mapping`: pares de los nombres de las columnas de la tabla de Hive y sus atributos correspondientes en la tabla de DynamoDB. Cada par tiene el formato nombre_de_columna_de_hive:nombre_de_atributo_de_dynamodb y los pares están separados entre sí por comas.

Tenga en cuenta lo siguiente:

- El nombre de la tabla de Hive no tiene que ser igual que el de la tabla de DynamoDB.
- Los nombres de las columnas de la tabla de Hive no tienen que ser iguales que los de la tabla de DynamoDB.
- La tabla especificada en `dynamodb.table.name` debe existir previamente en DynamoDB.
- En:`dynamodb.column.mapping`
  - Debe mapear los atributos del esquema de claves de la tabla de DynamoDB. Esto incluye la clave de partición y la clave de ordenación (si la hay).
  - No tiene que mapear los atributos que no son clave de la tabla de DynamoDB. Sin embargo, no aparecerá ningún dato de esos atributos cuando consulte la tabla de Hive.
  - Si los tipos de datos de una columna de la tabla de Hive y de un atributo de DynamoDB son incompatibles, aparecerá `NULL` en esas columnas cuando consulte la tabla de Hive.

> **Note**
>
> La instrucción `CREATE EXTERNAL TABLE` no lleva a cabo ninguna validación relativa a la cláusula `TBLPROPERTIES`. Los valores que proporcione para `dynamodb.table.name` y `dynamodb.column.mapping` solamente serán evaluados por la clase `DynamoDBStorageHandler` cuando se intente obtener acceso a la tabla.

Mapeos de tipos de datos

En la tabla siguiente se muestran los tipos de datos de DynamoDB y aquellos de Hive que son compatibles:

Tipo de dato de DynamoDB	Tipo de datos de Hive
Cadena	`STRING`
Número	`BIGINT` o `DOUBLE`
Binario	`BINARY`
String Set	`ARRAY<STRING>`
Number Set	`ARRAY<BIGINT>` o `ARRAY<DOUBLE>`

Tipo de dato de DynamoDB	Tipo de datos de Hive
Binary Set	`ARRAY<BINARY>`

> **Note**
>
> La clase `DynamoDBStorageHandler` no admite los siguientes tipos de datos de DynamoDB, por lo que estos no se pueden utilizar con `dynamodb.column.mapping`:
>
> - Asignación
> - Enumeración
> - Booleano
> - Nulo
>
> No obstante, si necesita trabajar con estos tipos de datos, puede crear una única entidad llamada `item` que represente todo el elemento de DynamoDB como un mapa de cadenas tanto para las claves como para los valores del mapa. Para obtener más información, consulte Copia de datos sin mapeo de columnas

Si desea mapear un atributo de DynamoDB del tipo Number (número), debe elegir un tipo de Hive apropiado:

- El tipo `BIGINT` de Hive es para enteros de 8 bytes con signo. Es igual que el tipo de datos `long` de Java.
- El tipo `DOUBLE` de Hive es para números de 8 bits con coma flotante de doble precisión. Es igual que el tipo `double` de Java.

Si tiene datos numéricos almacenados en DynamoDB cuya precisión es superior a la del tipo de datos de Hive que ha elegido, al obtener acceso a los datos de DynamoDB podría perderse precisión.

Si exporta datos del tipo Binary (binario) de DynamoDB a (Amazon S3) o HDFS, los datos se almacenarán como una cadena codificada en Base64. Si importa datos de Amazon S3 o HDFS al

tipo Binary (binario) de DynamoDB, debe asegurarse de que estén codificados como una cadena Base64.

Procesamiento de instrucciones de HiveQL

Hive es una aplicación que se ejecuta en Hadoop, que es un marco de trabajo orientado al procesamiento por lotes para ejecutar trabajos de MapReduce. Cuando emite una instrucción de HiveQL, Hive determina si puede devolver los resultados de forma inmediata o si se debe enviar un trabajo de MapReduce.

Por ejemplo, considere la tabla ddb_features (de Tutorial: Uso de Amazon DynamoDB y Apache Hive). La siguiente consulta de Hive imprime las abreviaturas de los estados y el número de cumbres de cada uno de ellos:

```
SELECT state_alpha, count(*)
FROM ddb_features
WHERE feature_class = 'Summit'
GROUP BY state_alpha;
```

Hive no devuelve los resultados de forma inmediata. En lugar de ello, envía un trabajo de MapReduce, que se procesa en el marco Hadoop. Hive espera hasta que se ha completado el trabajo para mostrar los resultados de la consulta:

```
AK  2
AL  2
AR  2
AZ  3
CA  7
CO  2
CT  2
ID  1
KS  1
ME  2
MI  1
MT  3
NC  1
NE  1
NM  1
NY  2
OR  5
PA  1
```

```
TN  1
TX  1
UT  4
VA  1
VT  2
WA  2
WY  3
Time taken: 8.753 seconds, Fetched: 25 row(s)
```

Monitorización y cancelación de trabajos

Cuando Hive lanza un trabajo de Hadoop, imprime el resultado de ese trabajo. El estado de realización del trabajo se actualiza a medida que este progresa. En algunos casos, el estado podría tardar mucho en actualizarse. Esto puede suceder cuando se consulta una tabla de DynamoDB de gran tamaño con un ajuste bajo de capacidad de lectura provisionada.

Si necesita cancelar el trabajo antes de que finalice, puede pulsar las teclas **`Ctrl+C`** en cualquier momento.

Consulta de datos en DynamoDB

En los siguientes ejemplos se muestran algunas formas de usar HiveQL para consultar los datos almacenados en DynamoDB.

Estos ejemplos se refieren a la tabla ddb_features del tutorial (Paso 5: copiar los datos a DynamoDB).

Temas

- Uso de funciones de agregación
- Uso de las cláusulas GROUP BY y HAVING
- Unión de dos tablas de DynamoDB
- Unión de tablas de orígenes diferentes

Uso de funciones de agregación

HiveQL proporciona funciones integradas para sintetizar los valores de los datos. Por ejemplo, puede utilizar la función `MAX` para hallar el valor máximo de una columna seleccionada. En el siguiente ejemplo se devuelve la altitud del accidente geográfico más alto del estado de Colorado.

```
SELECT MAX(elev_in_ft)
FROM ddb_features
WHERE state_alpha = 'CO';
```

Uso de las cláusulas GROUP BY y HAVING

Puede utilizar la cláusula `GROUP BY` para recopilar datos a través de varios registros. Esto se usa a menudo con una función de agregación, como `SUM`, `COUNT`, `MIN` o `MAX`. También puede usar la cláusula `HAVING` para descartar todos los resultados que no cumplan determinados criterios.

En el siguiente ejemplo se devuelve una lista con las cinco mayores altitudes de aquellos estados que tienen más de cinco accidentes geográficos contenidos en la tabla ddb_features.

```
SELECT state_alpha, max(elev_in_ft)
FROM ddb_features
GROUP BY state_alpha
HAVING count(*) >= 5;
```

Unión de dos tablas de DynamoDB

En el siguiente ejemplo se mapea otra tabla de Hive (east_coast_states) a una tabla de DynamoDB. La instrucción `SELECT` es una unión de estas dos tablas. La unión se calcula en el clúster y se devuelve. La unión no tiene lugar en DynamoDB.

Tomemos una tabla de DynamoDB denominada EastCoastStates que contiene los siguientes datos:

```
StateName       StateAbbrev

Maine           ME
New Hampshire   NH
Massachusetts   MA
Rhode Island    RI
Connecticut     CT
New York        NY
New Jersey      NJ
Delaware        DE
Maryland        MD
Virginia        VA
North Carolina  NC
South Carolina  SC
Georgia         GA
```

```
Florida         FL
```

Supongamos que la tabla está disponible como una tabla de Hive externa denominada east_coast_states:

```
CREATE EXTERNAL TABLE ddb_east_coast_states (state_name STRING, state_alpha STRING)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "EastCoastStates",
"dynamodb.column.mapping" = "state_name:StateName,state_alpha:StateAbbrev");
```

La unión siguiente devuelve los estados de la costa oriental de Estados Unidos que tienen al menos tres accidentes geográficos:

```
SELECT ecs.state_name, f.feature_class, COUNT(*)
FROM ddb_east_coast_states ecs
JOIN ddb_features f on ecs.state_alpha = f.state_alpha
GROUP BY ecs.state_name, f.feature_class
HAVING COUNT(*) >= 3;
```

Unión de tablas de orígenes diferentes

En el siguiente ejemplo, s3_east_coast_states es una tabla de Hive asociada a un archivo CSV almacenado en Amazon S3. La tabla ddb_features está asociada con datos de DynamoDB. En el ejemplo siguiente se unen estas dos tablas y se devuelven los accidentes geográficos de los estados cuyos nombres comienzan por "New".

```
create external table s3_east_coast_states (state_name STRING, state_alpha STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://bucketname/path/subpath/';
```

```
SELECT ecs.state_name, f.feature_name, f.feature_class
FROM s3_east_coast_states ecs
JOIN ddb_features f
ON ecs.state_alpha = f.state_alpha
WHERE ecs.state_name LIKE 'New%';
```

Copia de datos en y desde Amazon DynamoDB

En el Tutorial: Uso de Amazon DynamoDB y Apache Hive, hemos copiado datos de una tabla de Hive nativa a una tabla de DynamoDB externa y, a continuación, hemos consultado la tabla de

DynamoDB externa. La tabla es externa porque existe fuera de Hive. Aunque se elimine la tabla de Hive que está mapeada a ella, la tabla de DynamoDB no se verá afectada.

Hive es una solución excelente para copiar datos entre tablas de DynamoDB, buckets de Amazon S3, tablas de Hive nativas y Hadoop Distributed File System (HDFS). En esta sección se proporcionan ejemplos de estas operaciones.

Temas

- Copia de datos entre DynamoDB y una tabla nativa de Hive
- Copia de datos entre DynamoDB y Amazon S3
- Copia de datos entre DynamoDB y HDFS
- Uso de la compresión de datos
- Lectura de datos de caracteres UTF-8 no imprimibles

Copia de datos entre DynamoDB y una tabla nativa de Hive

Si tiene datos en una tabla de DynamoDB, puede copiarlos en una tabla de Hive nativa. Al hacerlo, obtendrá una instantánea de los datos en el momento de la copia.

Puede ser algo conveniente si tiene que realizar muchas consultas de HiveQL, pero no desea consumir capacidad de rendimiento aprovisionada de DynamoDB. Dado que los datos de la tabla de Hive nativa son una copia de los datos de DynamoDB, y no son datos "en tiempo real", en las consultas no cabe esperar que los datos estén actualizados.

> **Note**
>
> Los ejemplos de esta sección se han escrito partiendo del supuesto de que ya ha llevado a cabo los pasos del Tutorial: Uso de Amazon DynamoDB y Apache Hive y de que ya dispone de una tabla externa en DynamoDB llamada ddb_features.

Example Desde DynamoDB a una tabla nativa de Hive

Puede crear una tabla de Hive nativa y rellenarla con datos de ddb_features, así:

```
CREATE TABLE features_snapshot AS
SELECT * FROM ddb_features;
```

A continuación, puede actualizar los datos en cualquier momento:

```
INSERT OVERWRITE TABLE features_snapshot
SELECT * FROM ddb_features;
```

En estos ejemplos, la subconsulta `SELECT * FROM ddb_features` recuperará todos los datos de ddb_features. Si solamente desea copiar un subconjunto de los datos, puede usar una cláusula `WHERE` en la subconsulta.

En el siguiente ejemplo se crea una tabla de Hive nativa que únicamente contiene algunos de atributos de lagos y cumbres:

```
CREATE TABLE lakes_and_summits AS
SELECT feature_name, feature_class, state_alpha
FROM ddb_features
WHERE feature_class IN ('Lake','Summit');
```

Example Desde una tabla nativa de Hive a DynamoDB

La instrucción de HiveQL siguiente permite copiar los datos de la tabla de Hive nativa a ddb_features:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM features_snapshot;
```

Copia de datos entre DynamoDB y Amazon S3

Si tiene datos en una tabla de DynamoDB, puede usar Hive para copiarlos en un bucket de Amazon S3.

Puede ser interesante hacerlo si desea crear un archivo de datos en la tabla de DynamoDB. Por ejemplo, supongamos que tiene un entorno de pruebas en el cual necesita trabajar con un conjunto de datos de prueba de referencia en DynamoDB. Puede copiar los datos de referencia en un bucket de Amazon S3 y, a continuación, ejecutar las pruebas. A continuación, puede restablecer el entorno de pruebas restaurando los datos de referencia del bucket de Amazon S3 en DynamoDB.

Si ha realizado el Tutorial: Uso de Amazon DynamoDB y Apache Hive, ya dispone de un bucket de Amazon S3 que contiene los registros de Amazon EMR. Puede utilizar este bucket para los ejemplos de esta sección, si conoce su ruta raíz:

1. Abra la consola de Amazon EMR enhttps://console.aws.amazon.com/emr.

2. En Name, elija el clúster.
3. La URI aparece en Log URI bajo Configuration Details.
4. Anote la ruta raíz del bucket. La convención de nomenclatura es:

   `s3://aws-logs-`*`accountID`*`-`*`region`*

   donde *`accountID`* es el ID de su cuenta de AWS y region es la región de AWS del bucket.

> **Note**
>
> En estos ejemplos, utilizaremos una subruta contenida en el bucket, como en este ejemplo: `s3://aws-logs-123456789012-us-west-2/hive-test`

Los procedimientos siguientes se han escrito partiendo del supuesto de que ya ha llevado a cabo los pasos del tutorial y de que ya dispone de una tabla externa en DynamoDB llamada ddb_features.

Temas

- Copia de datos mediante el formato predeterminado de Hive
- Copia de datos con un formato especificado por el usuario
- Copia de datos sin mapeo de columnas
- Visualización de datos en Amazon S3

Copia de datos mediante el formato predeterminado de Hive

Example Desde DynamoDB a Amazon S3

Use una instrucción `INSERT OVERWRITE` para escribir directamente en Amazon S3.

```
INSERT OVERWRITE DIRECTORY 's3://aws-logs-123456789012-us-west-2/hive-test'
SELECT * FROM ddb_features;
```

El archivo de datos de Amazon S3 tiene este aspecto:

```
920709^ASoldiers Farewell Hill^ASummit^ANM^A32.3564729^A-108.33004616135
1178153^AJones Run^AStream^APA^A41.2120086^A-79.25920781260
253838^ASentinel Dome^ASummit^ACA^A37.7229821^A-119.584338133
264054^ANeversweet Gulch^AValley^ACA^A41.6565269^A-122.83614322900
```

```
115905^AChacaloochee Bay^ABay^AAL^A30.6979676^A-87.97388530
```

Cada campo se separa por un carácter SOH (inicio del encabezado, 0x01). En el archivo, SOH aparece como **^A**.

Example Desde Amazon S3 a DynamoDB

1. Cree una tabla externa que apunte a los datos sin formato de Amazon S3.

```
CREATE EXTERNAL TABLE s3_features_unformatted
    (feature_id       BIGINT,
    feature_name      STRING ,
    feature_class     STRING ,
    state_alpha       STRING,
    prim_lat_dec      DOUBLE ,
    prim_long_dec     DOUBLE ,
    elev_in_ft        BIGINT)
LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
```

2. Copie los datos en DynamoDB.

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM s3_features_unformatted;
```

Copia de datos con un formato especificado por el usuario

Si desea especificar su propio carácter separador de campos, puede crear una tabla externa que esté mapeada al bucket de Amazon S3. Puede utilizar esta técnica para crear archivos de datos con valores separados por comas (CSV).

Example Desde DynamoDB a Amazon S3

1. Cree una tabla de Hive externa mapeada a Amazon S3. Al hacerlo, asegúrese de que los tipos de datos sean coherentes con los de la tabla de DynamoDB externa.

```
CREATE EXTERNAL TABLE s3_features_csv
    (feature_id       BIGINT,
    feature_name      STRING,
    feature_class     STRING,
    state_alpha       STRING,
    prim_lat_dec      DOUBLE,
```

```
    prim_long_dec     DOUBLE,
    elev_in_ft        BIGINT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
```

2. Copie los datos de DynamoDB.

```
INSERT OVERWRITE TABLE s3_features_csv
SELECT * FROM ddb_features;
```

El archivo de datos de Amazon S3 tiene este aspecto:

```
920709,Soldiers Farewell Hill,Summit,NM,32.3564729,-108.3300461,6135
1178153,Jones Run,Stream,PA,41.2120086,-79.2592078,1260
253838,Sentinel Dome,Summit,CA,37.7229821,-119.58433,8133
264054,Neversweet Gulch,Valley,CA,41.6565269,-122.8361432,2900
115905,Chacaloochee Bay,Bay,AL,30.6979676,-87.9738853,0
```

Example Desde Amazon S3 a DynamoDB

Con una sola instrucción de HiveQL, puede rellenar la tabla de DynamoDB usando los datos de Amazon S3:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM s3_features_csv;
```

Copia de datos sin mapeo de columnas

Puede copiar datos de DynamoDB en formato sin procesar y escribirlos en Amazon S3 sin especificar ningún tipo de datos ni mapeo de columnas. Puede utilizar este método para crear un archivo comprimido de los datos de DynamoDB y almacenarlos en Amazon S3.

Example Desde DynamoDB a Amazon S3

1. Cree una tabla externa asociada con la tabla de DynamoDB. Esta instrucción de HiveQL no contiene ningún mapeo `dynamodb.column.mapping`.

```
CREATE EXTERNAL TABLE ddb_features_no_mapping
    (item MAP<STRING, STRING>)
```

```
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "Features");
```

2. Cree otra tabla externa asociada con el bucket de Amazon S3.

```
CREATE EXTERNAL TABLE s3_features_no_mapping
    (item MAP<STRING, STRING>)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
LOCATION 's3://aws-logs-123456789012-us-west-2/hive-test';
```

3. Copie los datos de DynamoDB a Amazon S3.

```
INSERT OVERWRITE TABLE s3_features_no_mapping
SELECT * FROM ddb_features_no_mapping;
```

El archivo de datos de Amazon S3 tiene este aspecto:

```
Name^C{"s":"Soldiers Farewell
 Hill"}^BState^C{"s":"NM"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"6135"}^BLatitude^C{"n":"32.
Name^C{"s":"Jones
 Run"}^BState^C{"s":"PA"}^BClass^C{"s":"Stream"}^BElevation^C{"n":"1260"}^BLatitude^C{"n":"41.2
Name^C{"s":"Sentinel
 Dome"}^BState^C{"s":"CA"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"8133"}^BLatitude^C{"n":"37.
Name^C{"s":"Neversweet
 Gulch"}^BState^C{"s":"CA"}^BClass^C{"s":"Valley"}^BElevation^C{"n":"2900"}^BLatitude^C{"n":"41
Name^C{"s":"Chacaloochee
 Bay"}^BState^C{"s":"AL"}^BClass^C{"s":"Bay"}^BElevation^C{"n":"0"}^BLatitude^C{"n":"30.6979676
```

Cada campo comienza por un carácter STX (inicio del texto, 0x02) y termina con un carácter ETX (final del texto, 0x03). En el archivo, STX aparece como **^B** y ETX aparece como **^C**.

Example Desde Amazon S3 a DynamoDB

Con una sola instrucción de HiveQL, puede rellenar la tabla de DynamoDB usando los datos de Amazon S3:

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM s3_features_no_mapping;
```

Visualización de datos en Amazon S3

Si usa SSH para conectarse al nodo líder, puede usar la AWS Command Line Interface (AWS CLI) para acceder a los datos que Hive escribió en Amazon S3.

Los siguientes pasos se han escrito partiendo del supuesto de que ha copiado los datos de DynamoDB a Amazon S3 por medio de uno de los procedimientos de esta sección.

1. Si tiene abierto el símbolo del sistema de Hive, salga y cambie al símbolo del sistema de Linux.

   ```
   hive> exit;
   ```

2. Muestre el contenido del directorio hive-test en el bucket de Amazon S3. (Aquí es donde Hive ha copiado los datos de DynamoDB).

   ```
   aws s3 ls s3://aws-logs-123456789012-us-west-2/hive-test/
   ```

   El aspecto de la respuesta debe ser parecido al siguiente:

   ```
   2016-11-01 23:19:54 81983 000000_0
   ```

   El nombre de archivo (000000_0) lo genera el sistema.

3. (Opcional) Puede copiar el archivo de datos de Amazon S3 al sistema de archivos local en el nodo líder. Después de hacerlo, puede utilizar las utilidades estándar de la línea de comandos de Linux para trabajar con los datos del archivo.

   ```
   aws s3 cp s3://aws-logs-123456789012-us-west-2/hive-test/000000_0 .
   ```

   El aspecto de la respuesta debe ser parecido al siguiente:

   ```
   download: s3://aws-logs-123456789012-us-west-2/hive-test/000000_0
   to ./000000_0
   ```

   > **Note**
   >
   > El sistema de archivos local del nodo líder tiene una capacidad limitada. No utilice este comando con archivos más grandes que el espacio disponible en el sistema de archivos local.

Copia de datos entre DynamoDB y HDFS

Si tiene datos en una tabla de DynamoDB, puede usar Hive para copiarlos a Hadoop Distributed File System (HDFS).

Puede ser conveniente si va a ejecutar un trabajo de MapReduce que requiere datos de DynamoDB. Si copia los datos de DynamoDB a HDFS, Hadoop puede procesarlos utilizando en paralelo todos los nodos disponibles en el clúster de Amazon EMR. Una vez que se haya completado el trabajo de MapReduce, puede escribir los resultados de HDFS a DDB.

En los ejemplos siguientes, Hive leerá y escribirá en el siguiente directorio de HDFS: `/user/hadoop/hive-test`

> **Note**
>
> Los ejemplos de esta sección se han escrito partiendo del supuesto de que ya ha llevado a cabo los pasos en Tutorial: Uso de Amazon DynamoDB y Apache Hive y de que ya dispone de una tabla externa en DynamoDB llamada ddb_features.

Temas

- Copia de datos mediante el formato predeterminado de Hive
- Copia de datos con un formato especificado por el usuario
- Copia de datos sin mapeo de columnas
- Acceso a los datos de HDFS

Copia de datos mediante el formato predeterminado de Hive

Example Desde DynamoDB a HDFS

Use una instrucción `INSERT OVERWRITE` para escribir directamente en HDFS.

```
INSERT OVERWRITE DIRECTORY 'hdfs:///user/hadoop/hive-test'
SELECT * FROM ddb_features;
```

El archivo de datos de HDFS tiene este aspecto:

```
920709^ASoldiers Farewell Hill^ASummit^ANM^A32.3564729^A-108.33004616135
1178153^AJones Run^AStream^APA^A41.2120086^A-79.25920781260
```

```
253838^ASentinel Dome^ASummit^ACA^A37.7229821^A-119.584338133
264054^ANeversweet Gulch^AValley^ACA^A41.6565269^A-122.83614322900
115905^AChacaloochee Bay^ABay^AAL^A30.6979676^A-87.97388530
```

Cada campo se separa por un carácter SOH (inicio del encabezado, 0x01). En el archivo, SOH aparece como **^A**.

Example Desde HDFS a DynamoDB

1. Cree una tabla externa mapeada a los datos sin formato de HDFS.

   ```
   CREATE EXTERNAL TABLE hdfs_features_unformatted
       (feature_id       BIGINT,
       feature_name      STRING ,
       feature_class     STRING ,
       state_alpha       STRING,
       prim_lat_dec      DOUBLE ,
       prim_long_dec     DOUBLE ,
       elev_in_ft        BIGINT)
   LOCATION 'hdfs:///user/hadoop/hive-test';
   ```

2. Copie los datos en DynamoDB.

   ```
   INSERT OVERWRITE TABLE ddb_features
   SELECT * FROM hdfs_features_unformatted;
   ```

Copia de datos con un formato especificado por el usuario

Si desea usar un carácter separador de campos distinto, puede crear una tabla externa que esté mapeada al directorio de HDFS. Puede utilizar esta técnica para crear archivos de datos con valores separados por comas (CSV).

Example Desde DynamoDB a HDFS

1. Cree una tabla de Hive externa mapeada a HDFS. Al hacerlo, asegúrese de que los tipos de datos sean coherentes con los de la tabla de DynamoDB externa.

   ```
   CREATE EXTERNAL TABLE hdfs_features_csv
       (feature_id       BIGINT,
       feature_name      STRING ,
       feature_class     STRING ,
   ```

```
    state_alpha       STRING,
    prim_lat_dec      DOUBLE ,
    prim_long_dec     DOUBLE ,
    elev_in_ft        BIGINT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 'hdfs:///user/hadoop/hive-test';
```

2. Copie los datos de DynamoDB.

```
INSERT OVERWRITE TABLE hdfs_features_csv
SELECT * FROM ddb_features;
```

El archivo de datos de HDFS tiene este aspecto:

```
920709,Soldiers Farewell Hill,Summit,NM,32.3564729,-108.3300461,6135
1178153,Jones Run,Stream,PA,41.2120086,-79.2592078,1260
253838,Sentinel Dome,Summit,CA,37.7229821,-119.58433,8133
264054,Neversweet Gulch,Valley,CA,41.6565269,-122.8361432,2900
115905,Chacaloochee Bay,Bay,AL,30.6979676,-87.9738853,0
```

Example Desde HDFS a DynamoDB

Con una sola instrucción de HiveQL, puede rellenar la tabla de DynamoDB usando los datos de HDFS:

```
INSERT OVERWRITE TABLE ddb_features
SELECT * FROM hdfs_features_csv;
```

Copia de datos sin mapeo de columnas

Puede copiar los datos de DynamoDB en formato sin procesar y escribirlos en HDFS sin especificar ningún tipo de datos ni mapeo de columnas. Puede utilizar este método para crear un archivo de los datos de DynamoDB y almacenarlos en HDFS.

> **Note**
>
> Si la tabla de DynamoDB contiene atributos de tipo Map (mapeo), List (lista), Boolean (booleano) o Null (nulo), entonces este es el único modo de usar Hive para copiar datos de DynamoDB a HDFS.

Example Desde DynamoDB a HDFS

1. Cree una tabla externa asociada con la tabla de DynamoDB. Esta instrucción de HiveQL no contiene ningún mapeo `dynamodb.column.mapping`.

```
CREATE EXTERNAL TABLE ddb_features_no_mapping
    (item MAP<STRING, STRING>)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES ("dynamodb.table.name" = "Features");
```

2. Cree otra tabla externa asociada con el directorio de HDFS.

```
CREATE EXTERNAL TABLE hdfs_features_no_mapping
    (item MAP<STRING, STRING>)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
LOCATION 'hdfs:///user/hadoop/hive-test';
```

3. Copie los datos de DynamoDB a HDFS.

```
INSERT OVERWRITE TABLE hdfs_features_no_mapping
SELECT * FROM ddb_features_no_mapping;
```

El archivo de datos de HDFS tiene este aspecto:

```
Name^C{"s":"Soldiers Farewell
 Hill"}^BState^C{"s":"NM"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"6135"}^BLatitude^C{"n":"32.
Name^C{"s":"Jones
 Run"}^BState^C{"s":"PA"}^BClass^C{"s":"Stream"}^BElevation^C{"n":"1260"}^BLatitude^C{"n":"41.2
Name^C{"s":"Sentinel
 Dome"}^BState^C{"s":"CA"}^BClass^C{"s":"Summit"}^BElevation^C{"n":"8133"}^BLatitude^C{"n":"37.
```

```
Name^C{"s":"Neversweet
 Gulch"}^BState^C{"s":"CA"}^BClass^C{"s":"Valley"}^BElevation^C{"n":"2900"}^BLatitude^C{"n":"41
Name^C{"s":"Chacaloochee
 Bay"}^BState^C{"s":"AL"}^BClass^C{"s":"Bay"}^BElevation^C{"n":"0"}^BLatitude^C{"n":"30.6979676
```

Cada campo comienza por un carácter STX (inicio del texto, 0x02) y termina con un carácter ETX (final del texto, 0x03). En el archivo, STX aparece como **^B** y ETX aparece como **^C**.

Example Desde HDFS a DynamoDB

Con una sola instrucción de HiveQL, puede rellenar la tabla de DynamoDB usando los datos de HDFS:

```
INSERT OVERWRITE TABLE ddb_features_no_mapping
SELECT * FROM hdfs_features_no_mapping;
```

Acceso a los datos de HDFS

HDFS es un sistema de archivos distribuido, accesible para todos los nodos del clúster de Amazon EMR. Si usa SSH para conectarse al nodo líder, puede usar las herramientas de línea de comandos para acceder a los datos que Hive escribió en HDFS.

HDFS no es lo mismo que el sistema de archivos local del nodo líder. No se puede trabajar con los archivos y directorios de HDFS mediante los comandos de Linux estándar (tales como `cat`, `cp`, `mv` o `rm`). En lugar de ello, estas tareas se llevan a cabo usando el comando `hadoop fs`.

Los pasos siguientes se han escrito partiendo del supuesto de que ha copiado los datos de DynamoDB a HDFS por medio de uno de los procedimientos de esta sección.

1. Si tiene abierto el símbolo del sistema de Hive, salga y cambie al símbolo del sistema de Linux.

   ```
   hive> exit;
   ```

2. Muestre el contenido del directorio /user/hadoop/hive-test en HDFS. (Aquí es donde Hive ha copiado los datos de DynamoDB).

   ```
   hadoop fs -ls /user/hadoop/hive-test
   ```

   El aspecto de la respuesta debe ser parecido al siguiente:

   ```
   Found 1 items
   ```

```
-rw-r--r-- 1 hadoop hadoop 29504 2016-06-08 23:40 /user/hadoop/hive-test/000000_0
```

El nombre de archivo (000000_0) lo genera el sistema.

3. Visualice el contenido del archivo :

```
hadoop fs -cat /user/hadoop/hive-test/000000_0
```

> **Note**
>
> En este ejemplo, el archivo es relativamente pequeño (aproximadamente, 29 KB). Tenga cuidado cuando utilice este comando con archivos que sean muy grandes o contengan caracteres no imprimibles.

4. (Opcional) Puede copiar el archivo de datos de HDFS al sistema de archivos local en el nodo maestro. Después de hacerlo, puede utilizar las utilidades estándar de la línea de comandos de Linux para trabajar con los datos del archivo.

```
hadoop fs -get /user/hadoop/hive-test/000000_0
```

Este comando no sobrescribirá el archivo.

> **Note**
>
> El sistema de archivos local del nodo líder tiene una capacidad limitada. No utilice este comando con archivos más grandes que el espacio disponible en el sistema de archivos local.

Uso de la compresión de datos

Cuando utilice Hive para copiar datos entre orígenes de datos diferentes, puede solicitar la compresión de datos sobre la marcha. Hive proporciona varios códecs de compresión. Puede elegir uno de ellos durante la sesión de Hive. Si lo hace, los datos se comprimirán en el formato especificado.

El siguiente ejemplo comprime los datos utilizando el algoritmo Lempel-Ziv-Oberhumer (LZO).

```
SET hive.exec.compress.output=true;
```

```
SET io.seqfile.compression.type=BLOCK;
SET mapred.output.compression.codec = com.hadoop.compression.lzo.LzopCodec;

CREATE EXTERNAL TABLE lzo_compression_table (line STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n'
LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE lzo_compression_table SELECT *
FROM hiveTableName;
```

El archivo resultante de Amazon S3 tendrá un nombre generado por el sistema con la extensión `.lzo` (por ejemplo, `8d436957-57ba-4af7-840c-96c2fc7bb6f5-000000.lzo`).

Los códecs de compresión disponibles son:

- `org.apache.hadoop.io.compress.GzipCodec`
- `org.apache.hadoop.io.compress.DefaultCodec`
- `com.hadoop.compression.lzo.LzoCodec`
- `com.hadoop.compression.lzo.LzopCodec`
- `org.apache.hadoop.io.compress.BZip2Codec`
- `org.apache.hadoop.io.compress.SnappyCodec`

Lectura de datos de caracteres UTF-8 no imprimibles

Para leer y escribir datos de caracteres UTF-8 no imprimibles, puede usar la cláusula `STORED AS SEQUENCEFILE` al crear una tabla de Hive. Un SequenceFile es un formato de archivo binario de Hadoop. Debe usar Hadoop para leer este archivo. En el siguiente ejemplo se muestra cómo exportar datos desde DynamoDB a Amazon S3. Puede utilizar esta funcionalidad para controlar caracteres codificados con UTF-8 no imprimibles.

```
CREATE EXTERNAL TABLE s3_export(a_col string, b_col bigint, c_col array<string>)
STORED AS SEQUENCEFILE
LOCATION 's3://bucketname/path/subpath/';

INSERT OVERWRITE TABLE s3_export SELECT *
FROM hiveTableName;
```

Ajuste del rendimiento

Al crear una tabla Hive externa mapeada a una tabla de DynamoDB, no está consumiendo ninguna capacidad de lectura o escritura de DynamoDB. Sin embargo, la actividad de lectura y escritura en la tabla de Hive (por ejemplo, `INSERT` o `SELECT`) se convierte directamente en operaciones de lectura y escritura en la tabla de DynamoDB subyacente.

Apache Hive en Amazon EMR implementa su propia lógica para balancear la carga de E/S en la tabla de DynamoDB y trata de minimizar la posibilidad de sobrepasar el rendimiento aprovisionado de la tabla. Al final de cada consulta de Hive, Amazon EMR devuelve métricas de tiempo de ejecución, incluido el número de veces que se ha superado el rendimiento aprovisionado. Puede utilizar esta información, junto con las métricas de CloudWatch sobre la tabla de DynamoDB, para mejorar el rendimiento en las solicitudes posteriores.

La consola de Amazon EMR proporciona herramientas de monitoreo básico para su clúster. Para obtener más información, consulte Ver y monitorear un clúster en la Guía de administración de Amazon EMR.

También puede monitorear el clúster y los trabajos de Hadoop mediante herramientas basadas en Web, tales como Hue, Ganglia y la interfaz web de Hadoop. Para obtener más información, consulte Ver la interfaz web alojada en los clústeres de Amazon EMR en la Guía de administración de Amazon EMR.

En esta sección se describen los pasos que puede llevar a cabo para ajustar el rendimiento de las operaciones de Hive en las tablas de DynamoDB externas.

Temas

- Rendimiento aprovisionado de DynamoDB
- Ajuste de mapeadores
- Temas adicionales

Rendimiento aprovisionado de DynamoDB

Cuando se emiten instrucciones de HiveQL para la tabla de DynamoDB externa, la clase `DynamoDBStorageHandler` realiza las solicitudes apropiadas de API de bajo nivel de DynamoDB, que consumen rendimiento aprovisionado. Si no hay suficiente capacidad de lectura o escritura en la tabla de DynamoDB, se aplica una limitación controlada a la solicitud, lo que da lugar a un

rendimiento lento de HiveQL. Por este motivo, debe asegurarse de que la tabla tenga suficiente capacidad de desempeño.

Por ejemplo, suponga que ha aprovisionado 100 unidades de capacidad de lectura para su tabla de DynamoDB. Esto permite leer 409 600 bytes por segundo (100 × 4 kb, que es el tamaño de la unidad de capacidad de lectura). Ahora, supongamos que la tabla contiene 20 GB de datos (a saber, 21 474 836 480 bytes) y que desea usar la instrucción `SELECT` para seleccionar todos los datos mediante HiveQL. A continuación se indica cómo calcular cuánto tardará aproximadamente la consulta en ejecutarse:

21 474 836 480 / 409 600 = 52 429 segundos = 14,56 horas

En esta situación, la tabla de DynamoDB es un cuello de botella. Agregar más nodos de Amazon EMR no servirá de ayuda, porque el rendimiento de Hive está limitado a tan solo 409 600 bytes por segundo. La única forma de reducir el tiempo necesario para ejecutar la instrucción `SELECT` es aumentar la capacidad de lectura aprovisionada de la tabla de DynamoDB.

Puede realizar un cálculo similar para calcular aproximadamente cuánto se tardaría en cargar los datos masivamente en una tabla de Hive externa mapeada a una tabla de DynamoDB. Determine el número total de unidades de capacidad de escritura necesarias por elemento (menos de 1 KB = 1, 1-2 KB = 2, etc.) y multiplíquelo por el número de elementos que se van a cargar. Obtendrá así el número de unidades de capacidad de escritura necesarias. Divida ese número entre el número de unidades de capacidad de escritura que se asignan por segundo. Obtendrá el número de segundos que se tardará en cargar la tabla.

Es conveniente monitorear periódicamente las métricas de CloudWatch correspondientes a la tabla. Para obtener una breve información general en la consola de DynamoDB, elija la tabla y, a continuación, elija la pestaña Metrics (Métricas). A partir de aquí, puede ver las unidades de capacidad de lectura y escritura consumidas y las solicitudes de lectura y escritura que han sido objeto de una limitación controlada.

Capacidad de lectura

Amazon EMR administra la carga de solicitudes en la tabla de DynamoDB de acuerdo con las configuraciones de rendimiento aprovisionado de la tabla. Sin embargo, si observa una cantidad elevada de mensajes de `ProvisionedThroughputExceeded` en el resultado del trabajo, puede ajustar la tasa de lectura predeterminada. Para ello, puede modificar la variable de configuración `dynamodb.throughput.read.percent`. Puede usar el comando `SET` para establecer esta variable en el símbolo del sistema de Hive:

```
SET dynamodb.throughput.read.percent=1.0;
```

Esta variable persiste únicamente durante la sesión de Hive actual. Si sale de Hive y vuelve a abrirlo más adelante, `dynamodb.throughput.read.percent` recuperará su valor predeterminado.

El valor de `dynamodb.throughput.read.percent` puede estar entre `0.1` y `1.5` de forma inclusiva. `0.5` representa la tasa de lectura predeterminada, lo que significa que Hive intentará consumir la mitad de la capacidad de lectura de la tabla. Si aumenta el valor por encima de `0.5`, Hive aumentará la tasa de solicitudes; al reducir el valor por debajo de `0.5`, disminuirá la tasa de solicitudes de lectura. (La tasa de lectura real varía, según diversos factores tales como el hecho de que exista o no una distribución uniforme de claves en la tabla de DynamoDB).

Si observa que Hive agota con frecuencia la capacidad de lectura aprovisionada de la tabla o si las solicitudes de lectura son objeto de la limitación controlada en demasiadas ocasiones, intente reducir `dynamodb.throughput.read.percent` por debajo de `0.5`. Si tiene capacidad de lectura suficiente en la tabla y desea aumentar la agilidad de las operaciones de HiveQL, puede establecer esta variable en un valor superior a `0.5`.

Capacidad de escritura

Amazon EMR administra la carga de solicitudes en la tabla de DynamoDB de acuerdo con las configuraciones de rendimiento aprovisionado de la tabla. Sin embargo, si observa una cantidad elevada de mensajes de `ProvisionedThroughputExceeded` en el resultado del trabajo, puede ajustar la tasa de escritura predeterminada. Para ello, puede modificar la variable de configuración `dynamodb.throughput.write.percent`. Puede usar el comando `SET` para establecer esta variable en el símbolo del sistema de Hive:

```
SET dynamodb.throughput.write.percent=1.0;
```

Esta variable persiste únicamente durante la sesión de Hive actual. Si sale de Hive y vuelve a abrirlo más adelante, `dynamodb.throughput.write.percent` recuperará su valor predeterminado.

El valor de `dynamodb.throughput.write.percent` puede estar entre `0.1` y `1.5` de forma inclusiva. `0.5` representa la tasa de escritura predeterminada, lo que significa que Hive intentará consumir la mitad de la capacidad de escritura de la tabla. Si aumenta el valor por encima de `0.5`, Hive aumentará la tasa de solicitudes; al reducir el valor por debajo de `0.5`, disminuirá la tasa de solicitudes de escritura. (La tasa de escritura real varía en función de diversos factores, tales como el hecho de que exista o no una distribución uniforme de claves en la tabla de DynamoDB).

Si observa que Hive agota con frecuencia la capacidad de escritura aprovisionada de la tabla, o si las solicitudes de escritura son objeto de la limitación controlada en demasiados casos, pruebe a reducir `dynamodb.throughput.write.percent` por debajo de `0.5`. Si tiene capacidad suficiente en la tabla y desea aumentar la agilidad de las operaciones de HiveQL, puede establecer esta variable en un valor superior a `0.5`.

Al escribir datos en DynamoDB utilizando Hive, debe asegurarse de que el número de unidades de capacidad de escritura sea mayor que el número de mapeadores en el cluster. Por ejemplo, tomemos un clúster de Amazon EMR que consta de 10 nodos m1.xlarge. El tipo de nodo m1.xlarge proporciona 8 tareas de mapeador, de modo que el clúster tendría un total de 80mapeadores (10 × 8). Si la tabla de DynamoDB tiene menos de 80 unidades de capacidad de escritura, entonces una operación de escritura de Hive podría consumir todo el rendimiento de escritura de dicha tabla.

Para determinar el número de mapeadores de los tipos de nodos de Amazon EMR, consulte Configuración de la tarea en la Guía para desarrolladores de Amazon EMR.

Para obtener más información sobre los mapeadores, consulte Ajuste de mapeadores.

Ajuste de mapeadores

Cuando Hive lanza un trabajo de Hadoop, el trabajo se procesa por una o varias tareas de mapeador. Suponiendo que su tabla de DynamoDB tenga suficiente capacidad de rendimiento, puede modificar el número de mapeadores del clúster, lo que podría mejorar el rendimiento.

> **Note**
>
> El número de tareas de mapeador que se usan en un trabajo de Hadoop se ve influenciado por las divisiones de entrada que Hadoop utiliza para subdividir los datos en bloques lógicos. Si Hadoop no realiza suficientes divisiones de entrada, entonces las operaciones de escritura podrían no consumir todo el rendimiento de escritura disponible en la tabla de DynamoDB.

Aumento del número de mapeadores

Cada mapeador de un Amazon EMR tiene una tasa de lectura máxima de 1 MiB por segundo. El número de mapeadores del clúster depende del tamaño de los nodos que este contiene. (Para obtener información sobre los tamaños de los nodos y el número de mapeadores por nodo, consulte Configuración de la tarea en la Guía de desarrolladores de Amazon EMR).

Si la tabla de DynamoDB posee gran capacidad de rendimiento para las lecturas, puede probar a aumentar el número de mapeadores de una de las siguientes formas:

- Aumente el tamaño de los nodos del clúster. Por ejemplo, si el clúster utiliza m1.large nodos (tres mapeadores por nodo), puede probar a actualizarlos a nodos m1.xlarge (ocho mapeadores por nodo).
- Aumente el número de nodos del clúster. Por ejemplo, si tiene un clúster de tres nodos de tipo m1.xlarge, dispone de un total de 24 mapeadores. Si duplicara el tamaño del clúster, con el mismo tipo de nodo, tendría 48 mapeadores.

Puede utilizar la Consola de administración de AWS para administrar el tamaño o el número de nodos del clúster. Puede que tenga que reiniciar el clúster para que surtan efecto estos cambios.

Otra forma de aumentar el número de mapeadores consiste en modificar el parámetro de configuración `mapred.tasktracker.map.tasks.maximum` de Hadoop. (Se trata de un parámetro de Hadoop, no de Hive. No puede modificarlo de forma interactiva desde el símbolo del sistema). Si aumenta el valor de `mapred.tasktracker.map.tasks.maximum`, puede incrementar el número de mapeadores sin aumentar el número ni el tamaño de los nodos. Sin embargo, es posible que los nodos del clúster se queden sin memoria si establece un valor demasiado elevado.

Configura el valor de `mapred.tasktracker.map.tasks.maximum` como acción de arranque la primera vez que lanza el clúster de Amazon EMR. Para obtener más información, consulte (Opcional) Crear acción de arranque para instalar software adicional en la Guía de administración de Amazon EMR.

Reducción del número de mapeadores

Si utiliza la instrucción `SELECT` para seleccionar datos de una tabla de Hive externa mapeada a DynamoDB, el trabajo de Hadoop puede utilizar tantas tareas como sea necesario, hasta el número máximo de mapeadores del clúster. En esta situación, es posible que una consulta de Hive que tarde bastante en ejecutarse consuma toda la capacidad de lectura aprovisionada de la tabla de DynamoDB, lo que afectaría negativamente a otros usuarios.

Puede usar el parámetro `dynamodb.max.map.tasks` para establecer un límite superior para las tareas de mapeo:

```
SET dynamodb.max.map.tasks=1
```

Este valor debe ser igual o superior a 1. Cuando Hive procese la consulta, el trabajo de Hadoop resultante no usará más `dynamodb.max.map.tasks` que las indicadas cuando realice lecturas en la tabla de DynamoDB.

Temas adicionales

A continuación se muestran otras maneras de ajustar las aplicaciones que utilizan Hive para acceder a DynamoDB.

Retry duration

De forma predeterminada, Hive vuelve a ejecutar un trabajo de Hadoop si este no devuelve resultados de DynamoDB en un plazo de dos minutos. Puede ajustar este intervalo modificando el parámetro `dynamodb.retry.duration`:

```
SET dynamodb.retry.duration=2;
```

El valor debe ser un número entero distinto de cero que represente el número de minutos del intervalo de reintento. El valor predeterminado de `dynamodb.retry.duration` es 2 (minutos).

Solicitudes de datos paralelas

Varias solicitudes de datos a una única tabla, ya sean de más de un usuario o de más de una aplicación, podrían agotar el desempeño de lectura provisionado y ralentizar el desempeño.

Duración del proceso

La consistencia de datos en DynamoDB depende del orden de las operaciones de lectura y escritura en cada nodo. Aunque haya una consulta de Hive en curso, otra aplicación podría cargar nuevos datos en la tabla de DynamoDB o modificar o eliminar datos existentes. En este caso, los resultados de la consulta de Hive podrían no reflejar los cambios realizados en los datos mientras se ejecutaba la consulta.

Tiempo de solicitud

Programar las consultas de Hive que acceden a una tabla de DynamoDB de modo que se lleven a cabo en aquellos momentos en que la demanda de esa tabla de DynamoDB es más baja, mejora el rendimiento. Por ejemplo, si la mayoría de los usuarios de la aplicación viven en San Francisco, podría elegir exportar los datos diarios a las 4.00 h PST, cuando la mayoría de los usuarios duerme y no actualizan los registros de la base de datos de DynamoDB.

Integración de DynamoDB con Amazon S3

Las capacidades de importación y exportación de Amazon DynamoDB proporcionan una forma sencilla y eficaz de mover datos entre las tablas de Amazon S3 y DynamoDB sin tener que escribir código.

Las características de importación y exportación de DynamoDB le ayudan a mover, transformar y copiar cuentas de tablas de DynamoDB. Puede importar desde los orígenes de S3 y puede exportar los datos de las tablas de DynamoDB a Amazon S3 y utilizar servicios de AWS como Athena, Amazon SageMaker AI y AWS Lake Formation para analizar los datos y extraer información procesable. También puede importar datos directamente a nuevas tablas de DynamoDB para crear nuevas aplicaciones con un rendimiento de un milisegundo a escala, facilitar el uso compartido de datos entre tablas y cuentas, y simplificar sus planes de recuperación de desastres y continuidad empresarial.

Temas

- Importación de datos de DynamoDB desde Amazon S3: cómo funciona
- Exportación de datos de DynamoDB a Amazon S3: cómo funciona

Importación de datos de DynamoDB desde Amazon S3: cómo funciona

Para importar datos a DynamoDB, estos deben estar en un bucket de Amazon S3 en formato CSV, JSON de DynamoDB o Amazon Ion. Los datos pueden comprimirse en formato ZSTD o GZIP, o pueden importarse directamente sin comprimir. Los datos de origen pueden ser un único objeto de Amazon S3 o varios objetos de Amazon S3 que utilicen el mismo prefijo.

Sus datos se importarán a una nueva tabla de DynamoDB, que se creará cuando inicie la solicitud de importación. Puede crear esta tabla con índices secundarios y, a continuación, consultar y actualizar los datos en todos los índices primarios y secundarios en cuanto se complete la importación. También puede agregar una réplica de la tabla global una vez finalizada la importación.

> **Note**
>
> Durante el proceso de importación de Amazon S3, DynamoDB crea una nueva tabla de destino a la que se importará. Esta característica no admite actualmente la importación en tablas existentes.

La importación desde Amazon S3 no consume capacidad de escritura en la nueva tabla, por lo que no es necesario aprovisionar ninguna capacidad adicional para importar datos a DynamoDB. El precio de la importación de datos se basa en el tamaño sin comprimir de los datos de origen en Amazon S3, que se procesan como resultado de la importación. Los elementos que se procesan pero que no se cargan en la tabla debido al formato u otras incoherencias en los datos de origen también se facturan como parte del proceso de importación. Consulte Precios de Amazon DynamoDB para obtener más detalles.

Puede importar datos de un bucket de Amazon S3 que pertenece a otra cuenta si tiene los permisos correctos para leer de ese bucket específico. La nueva tabla también puede estar en una región diferente del bucket de Amazon S3 de origen. Para obtener más información, consulte Configuración y permisos de Amazon Simple Storage Service.

Los tiempos de importación están directamente relacionados con las características de sus datos en Amazon S3. Esto incluye el tamaño de los datos, el formato de los datos, el esquema de compresión, la uniformidad de la distribución de los datos, el número de objetos de Amazon S3 y otras variables relacionadas. En concreto, los conjuntos de datos con claves distribuidas de un modo uniforme serán más rápidos de importar que los conjuntos de datos sesgados. Por ejemplo, si la clave del índice secundario utiliza el mes del año para la partición y todos sus datos son del mes de diciembre, la importación de estos datos puede tardar bastante más.

Se espera que los atributos asociados a las claves sean únicos en la tabla base. Si alguna clave no es única, la importación sobrescribirá los elementos asociados hasta que solo quede la última sobrescritura. Por ejemplo, si la clave principal es el mes y se establecen varios elementos en el mes de septiembre, cada nuevo elemento sobrescribirá los elementos escritos anteriormente y solo quedará un elemento con la clave principal de “mes” establecido en septiembre. En estos casos, el número de elementos procesados en la descripción de la tabla de importación no coincidirá con el número de elementos de la tabla de destino.

AWS CloudTrail registra todas las acciones de la consola y de la API para la importación de tablas. Para obtener más información, consulte Registrar las operaciones de DynamoDB mediante AWS CloudTrail.

El siguiente vídeo es una introducción a la importación directa desde Amazon S3 a DynamoDB.

Importación desde Amazon S3

Temas

- Solicitud de una importación de tabla en DynamoDB

- Formatos de importación de Amazon S3 para DynamoDB
- Cuotas de formato de importación y validación
- Prácticas recomendadas para importar de Amazon S3 a DynamoDB

Solicitud de una importación de tabla en DynamoDB

La importación de DynamoDB le permite importar datos de un bucket de Amazon S3 a una nueva tabla de DynamoDB. Puede solicitar una importación de tabla mediante la consola de DynamoDB, la CLI, CloudFormation o la API de DynamoDB.

Si quiere utilizar la AWS CLI, primero tendrá que configurarla. Para obtener más información, consulte Acceso a DynamoDB.

> **Note**
>
> - La característica de importación de tabla interactúa con varios servicios de AWS diferentes, como Amazon S3 y CloudWatch. Antes de iniciar una importación, asegúrese de que el usuario o rol que invoca las API de importación tiene permisos para todos los servicios y recursos de los que depende la característica.
> - No modifique los objetos de Amazon S3 mientras la importación esté en curso, ya que esto puede hacer que se produzca un error o que se cancele la operación.
>
> Para obtener más información sobre los errores y la resolución de problemas, consulte Cuotas de formato de importación y validación

Temas

- Configuración de permisos de IAM
- Solicitud de una importación mediante la Consola de administración de AWS
- Obtención de detalles sobre importaciones pasadas en la Consola de administración de AWS
- Solicitud de una importación mediante la AWS CLI
- Obtención de detalles sobre importaciones pasadas en la AWS CLI

Configuración de permisos de IAM

Puede importar datos de cualquier bucket de Amazon S3 del que tenga permiso para leer. El bucket de origen no tiene por qué estar en la misma región o tener el mismo propietario que la tabla fuente. Su AWS Identity and Access Management (IAM) debe incluir las acciones pertinentes en el bucket de Amazon S3 de origen y los permisos de CloudWatch necesarios para proporcionar información de depuración. A continuación, se muestra una política de ejemplo.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDynamoDBImportAction",
      "Effect": "Allow",
      "Action": [
        "dynamodb:ImportTable",
        "dynamodb:DescribeImport"
      ],
      "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table*"
    },
    {
      "Sid": "AllowS3Access",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::your-bucket/*",
        "arn:aws:s3:::your-bucket"
      ]
    },
    {
      "Sid": "AllowCloudwatchAccess",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
```

```
        "logs:PutLogEvents",
        "logs:PutRetentionPolicy"
      ],
      "Resource": "arn:aws:logs:us-east-1:111122223333:log-group/aws-dynamodb/*"
    },
    {
      "Sid": "AllowDynamoDBListImports",
      "Effect": "Allow",
      "Action": "dynamodb:ListImports",
      "Resource": "*"
    }
  ]
}
```

Permisos de Amazon S3

Al iniciar una importación en un origen de bucket de Amazon S3 que es propiedad de otra cuenta, asegúrese de que el rol o el usuario tiene acceso a los objetos de Amazon S3. Puede comprobarlo si ejecuta un comando `GetObject` de Amazon S3 y utiliza las credenciales. Al utilizar la API, el parámetro del propietario del bucket de Amazon S3 tiene como valor predeterminado el ID de la cuenta del usuario actual. Para las importaciones entre cuentas, asegúrese de que este parámetro se rellena correctamente con el ID de la cuenta del propietario del bucket. El siguiente código es un ejemplo de política de bucket de Amazon S3 en la cuenta de origen.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {"Sid": "ExampleStatement",
            "Effect": "Allow",
            "Principal": {"AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
```

```
}
```

AWS Key Management Service

Al crear la nueva tabla para la importación, si selecciona una clave de cifrado en reposo que no sea propiedad de DynamoDB, deberá proporcionar los permisos de AWS KMS necesarios para utilizar una tabla de DynamoDB cifrada con claves administradas por el cliente. Para obtener más información, consulte Autorización del uso de su clave de AWS KMS. Si los objetos de Amazon S3 están cifrados con KMS en el servidor (SSE-KMS), asegúrese de que el rol o el usuario que inicia la importación tiene acceso a descifrar mediante la clave de AWS KMS. Esta característica no es compatible con los objetos de Amazon S3 cifrados con claves de cifrado proporcionadas por el cliente (SSE-C).

Permisos de CloudWatch

El rol o usuario que inicia la importación necesitará permisos de creación y administración para el grupo de registros y los flujos de registros asociados a la importación.

Solicitud de una importación mediante la Consola de administración de AWS

El siguiente ejemplo demuestra cómo utilizar la consola de DynamoDB para importar datos existentes a una nueva tabla llamada `MusicCollection`.

Para solicitar una importación de tabla

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación del lado izquierdo de la consola, elija Import from S3 (Importar desde S3).
3. En la página que aparece, seleccione Import from S3 (Importar desde S3).
4. Elija Import from S3 (Importar desde S3).
5. En URL de origen de S3, introduzca la URL de origen de Amazon S3.

   Si es propietario del bucket de origen, elija Examinar S3 para buscarlo. También puede introducir la URL del bucket en el siguiente formato: `s3://bucket/prefix`. `prefix` es un prefijo de clave de Amazon S3. Es el nombre del objeto de Amazon S3 que desea importar o el prefijo de la clave compartida por todos los objetos de Amazon S3 que desea importar.

> **Note**
>
> No puede utilizar el mismo prefijo que su solicitud de exportación de DynamoDB. La característica de exportación crea una estructura de carpetas y archivos de manifiesto para todas las exportaciones. Si usa la misma ruta de Amazon S3, se producirá un error. En su lugar, dirija la importación a la carpeta que contiene los datos de esa exportación específica. El formato de la ruta correcta en este caso será `s3://bucket/prefix/AWSDynamoDB/<XXXXXXXX-XXXXXX>/Data/`, dónde `XXXXXXXX-XXXXXX` es el identificador de exportación. Puede encontrar el identificador de exportación en el ARN de exportación, que tiene el siguiente formato: `arn:aws:dynamodb:<Region>:<AccountID>:table/<TableName>/export/<XXXXXXXX-XXXXXX>`. Por ejemplo, `arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/export/01234567890123-a1b2c3d4`.

6. Especifique si es S3 bucket owner (Propietario del bucket de S3). Si el bucket de origen pertenece a otra cuenta, seleccione Una cuenta de AWS diferente. A continuación, ingrese el ID de la cuenta del propietario del bucket.
7. En Compresión del archivo de importación, seleccione Sin compresión, GZIP o ZSTD, según corresponda.
8. Seleccione el formato de archivo de importación adecuado. Las opciones son DynamoDB JSON, Amazon Ion o CSV. Si selecciona CSV, tendrá dos opciones adicionales: CSV header (Encabezado de CSV) y CSV delimiter character (Carácter delimitador de CSV).

   En CSV header (Encabezado CSV), elija si el encabezado se tomará de la primera línea del archivo o si se personalizará. Si selecciona Customize your headers (Personalizar los encabezados), puede especificar los valores de encabezado con los que desee hacer la importación. En los encabezados de CSV especificados por este método se distingue entre mayúsculas y minúsculas y se espera que contengan las claves de la tabla de destino.

   En CSV delimiter character (Carácter delimitador de CSV), se establece el carácter que separará los elementos. De forma predeterminada, está seleccionada la coma. Si selecciona Custom delimiter character (Carácter delimitador personalizado), el delimitador debe coincidir con el patrón de expresiones regex: `[,;:|\t ]`.
9. Seleccione el botón Next (Siguiente) y seleccione las opciones de la nueva tabla que se creará para almacenar sus datos.

> **Note**
>
> La clave principal y la clave de clasificación deben coincidir con los atributos del archivo; de lo contrario, se producirá un error en la importación. En los atributos se distingue entre mayúsculas y minúsculas.

10. Vuelva a seleccionar Next (Siguiente) para revisar las opciones de importación y, a continuación, haga clic en Import (Importar) para iniciar la tarea de importación. Primero verá la nueva tabla en "Tables" (Tablas) con el estado "Creating" (Creando). En este momento no se puede acceder a la tabla.
11. Una vez completada la importación, el estado se mostrará como "Active" (Activo) y podrá empezar a utilizar la tabla.

Obtención de detalles sobre importaciones pasadas en la Consola de administración de AWS

Puede encontrar información sobre las tareas de importación que ha ejecutado en el pasado si hace clic en Import from S3 (Importar desde S3) en la barra de navegación lateral y, a continuación, seleccione la pestaña Imports (Importaciones). El panel de importación contiene una lista de todas las importaciones que ha creado en los últimos 90 días. Al seleccionar el ARN de una tarea enumerada en la pestaña Imports (Importaciones), se recuperará información sobre esa importación, incluida la configuración avanzada que haya elegido.

Solicitud de una importación mediante la AWS CLI

El siguiente ejemplo importa datos con formato CSV desde un bucket de S3 llamado bucket con un prefijo de "prefix" a una nueva tabla llamada "target-table".

```
aws dynamodb import-table --s3-bucket-source S3Bucket=bucket,S3KeyPrefix=prefix \
            --input-format CSV --table-creation-parameters '{"TableName":"target-
table","KeySchema":  \
            [{"AttributeName":"hk","KeyType":"HASH"}],"AttributeDefinitions":
[{"AttributeName":"hk","AttributeType":"S"}],"BillingMode":"PAY_PER_REQUEST"}' \
            --input-format-options '{"Csv": {"HeaderList": ["hk", "title", "artist",
 "year_of_release"], "Delimiter": ";"}}'
```

> **Note**
>
> Si elige cifrar la importación mediante una clave protegida por AWS Key Management Service (AWS KMS), la clave debe estar en la misma región que el bucket de Amazon S3 de destino.

Obtención de detalles sobre importaciones pasadas en la AWS CLI

Puede encontrar información acerca de las tareas de importación que ha ejecutado en el pasado mediante el comando `list-imports`. Este comando devuelve una lista de todas las importaciones que ha creado en los últimos 90 días. Tenga en cuenta que, aunque los metadatos de la tarea de importación caducan a los 90 días y los trabajos más antiguos ya no se encuentran en esta lista, DynamoDB no elimina ninguno de los objetos de su bucket de Amazon S3 ni la tabla creada durante la importación.

```
aws dynamodb list-imports
```

Para recuperar información detallada sobre una tarea de importación específica, incluida la configuración avanzada, utilice el comando `describe-import`.

```
aws dynamodb describe-import \
    --import-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/exp
```

Formatos de importación de Amazon S3 para DynamoDB

DynamoDB puede importar datos en tres formatos: CSV, DynamoDB JSON y Amazon Ion.

Temas

- CSV
- DynamoDB Json
- Amazon Ion

CSV

Un archivo en formato CSV consta de varios elementos delimitados por saltos de línea. De forma predeterminada, DynamoDB interpreta la primera línea de un archivo de importación como

encabezado y espera que las columnas estén delimitadas por comas. También puede definir los encabezados que se aplicarán, siempre que coincidan con el número de columnas del archivo. Si define los encabezados de forma explícita, la primera línea del archivo se importará como valores.

> **Note**
>
> Al importar desde archivos CSV, todas las columnas, excepto el rango de hash y las claves de la tabla base y los índices secundarios, se importan como cadenas de DynamoDB.

Aplicación de la secuencia de escape a las comillas dobles

A los caracteres entre comillas dobles que haya en el archivo CSV se les debe aplicar la secuencia de escape. Si no tienen aplicada la secuencia de escape, como en el siguiente ejemplo, no se realizará la importación:

```
id,value
"123",Women's Full "Length" Dress
```

Esta misma importación se realizará correctamente si las comillas tienen aplicada la secuencia de escape con dos conjuntos de comillas dobles:

```
id,value
"""123""","Women's Full ""Length"" Dress"
```

Una vez que se haya aplicado correctamente la secuencia de escape al texto y se haya importado, aparecerá tal y como estaba en el archivo CSV original:

```
id,value
"123",Women's Full "Length" Dress
```

DynamoDB Json

Un archivo en formato DynamoDB JSON puede constar de varios objetos Item. Cada objeto individual está en formato JSON serializado estándar de DynamoDB y los saltos de línea se utilizan como delimitadores de elementos. Como característica adicional, las exportaciones desde un momento dado se admiten como origen de importación predeterminado.

> **Note**
>
> Las líneas nuevas se utilizan como delimitadores de elementos para un archivo en formato JSON de DynamoDB y no deben utilizarse dentro de un objeto de elemento.

```
{"Item": {"Authors": {"SS": ["Author1", "Author2"]}, "Dimensions": {"S": "8.5 x 11.0 x
 1.5"}, "ISBN": {"S": "333-3333333333"}, "Id": {"N": "103"}, "InPublication": {"BOOL":
 false}, "PageCount": {"N": "600"}, "Price": {"N": "2000"}, "ProductCategory": {"S":
 "Book"}, "Title": {"S": "Book 103 Title"}}}
{"Item": {"Authors": {"SS": ["Author1", "Author2"]}, "Dimensions": {"S": "8.5 x 11.0 x
 1.5"}, "ISBN": {"S": "444-444444444"}, "Id": {"N": "104"}, "InPublication": {"BOOL":
 false}, "PageCount": {"N": "600"}, "Price": {"N": "2000"}, "ProductCategory": {"S":
 "Book"}, "Title": {"S": "Book 104 Title"}}}
{"Item": {"Authors": {"SS": ["Author1", "Author2"]}, "Dimensions": {"S": "8.5 x 11.0 x
 1.5"}, "ISBN": {"S": "555-5555555555"}, "Id": {"N": "105"}, "InPublication": {"BOOL":
 false}, "PageCount": {"N": "600"}, "Price": {"N": "2000"}, "ProductCategory": {"S":
 "Book"}, "Title": {"S": "Book 105 Title"}}}
```

Amazon Ion

Amazon Ion es un formato de serialización de datos jerárquico, autodescriptivo y altamente digitado, creado para abordar los desafíos rápidos de desarrollo, desacoplamiento y eficiencia que surgen todos los días mientras se diseñan arquitecturas a gran escala orientadas a servicios.

Cuando se importan datos en formato Ion, los tipos de datos de Ion se asignan a los tipos de datos de DynamoDB en la nueva tabla de DynamoDB.

S. No.	Conversión de tipos de datos de Ion a DynamoDB	B
1	`Ion Data Type`	`DynamoDB Representation`
2	`string`	`String (s)`
3	`bool`	`Boolean (BOOL)`
4	`decimal`	`Number (N)`

S. No.	Conversión de tipos de datos de Ion a DynamoDB	B
5	`blob`	`Binary (B)`
6	`list (with type annotation $dynamodb_SS, $dynamodb_NS, or $dynamodb_BS)`	`Set (SS, NS, BS)`
7	`list`	`List`
8	`struct`	`Map`

Los elementos de un archivo de Ion están delimitados por saltos de línea. Cada línea comienza con un marcador de versión Ion, seguido de un elemento en formato Ion.

> **Note**
>
> En el siguiente ejemplo, hemos dado formato a los elementos de un archivo con formato Ion en varias líneas para mejorar la legibilidad.

```
$ion_1_0
[
  {
    Item:{
      Authors:$dynamodb_SS::["Author1","Author2"],
      Dimensions:"8.5 x 11.0 x 1.5",
      ISBN:"333-3333333333",
      Id:103.,
      InPublication:false,
      PageCount:6d2,
      Price:2d3,
      ProductCategory:"Book",
      Title:"Book 103 Title"
    }
  },
  {
    Item:{
```

```
      Authors:$dynamodb_SS::["Author1","Author2"],
      Dimensions:"8.5 x 11.0 x 1.5",
      ISBN:"444-4444444444",
      Id:104.,
      InPublication:false,
      PageCount:6d2,
      Price:2d3,
      ProductCategory:"Book",
      Title:"Book 104 Title"
    }
  },
  {
    Item:{
      Authors:$dynamodb_SS::["Author1","Author2"],
      Dimensions:"8.5 x 11.0 x 1.5",
      ISBN:"555-5555555555",
      Id:105.,
      InPublication:false,
      PageCount:6d2,
      Price:2d3,
      ProductCategory:"Book",
      Title:"Book 105 Title"
    }
  }
]
```

Cuotas de formato de importación y validación

Cuotas de importación

La importación de DynamoDB desde Amazon S3 puede admitir hasta 50 trabajos de importación simultáneos con un tamaño total de objetos de origen de importación de 15 TB a la vez en las regiones us-east-1, us-west-2 y eu-west-1. En todas las demás regiones, se admiten hasta 50 tareas de importación simultáneas con un tamaño total de 1 TB. Cada trabajo de importación admite un máximo de 50 000 objetos de Amazon S3 en todas las regiones. Estas cuotas predeterminadas se aplican a todas las cuentas. Si cree que necesita revisar estas cuotas, contacte con su equipo de cuentas y se estudiará cada caso. Para obtener más información sobre los límites de DynamoDB, consulte Service Quotas.

Errores de validación

Durante el proceso de importación, DynamoDB puede encontrar errores al analizar los datos. Para cada error, DynamoDB emite un registro de CloudWatch y mantiene un recuento del número total de errores encontrados. Si el propio objeto de Amazon S3 tiene un formato incorrecto o si su contenido no puede formar un elemento de DynamoDB, podemos omitir el procesamiento de la parte restante del objeto.

> **Note**
>
> Si el origen de datos de Amazon S3 tiene varios elementos que comparten la misma clave, los elementos se sobrescribirán hasta que quede uno. Puede parecer que se hubiera importado un elemento y se hubieran ignorado los demás. Los elementos duplicados se sobrescribirán en orden aleatorio, no se contarán como errores y no se emitirán a los registros de CloudWatch.
> Una vez finalizada la importación, podrá ver el recuento total de elementos importados, el recuento total de errores y el recuento total de elementos procesados. Para seguir resolviendo problemas, también puede verificar el tamaño total de los elementos importados y el tamaño total de los datos procesados.

Hay tres categorías de errores de importación: errores de validación de la API, errores de validación de datos y errores de configuración.

Errores de validación de la API

Los errores de validación de la API son errores en el nivel de elementos de la API de sincronización. Las causas más comunes son los problemas de permisos, la falta de parámetros necesarios y los errores de validación de los parámetros. Los detalles del motivo del error de la llamada a la API se encuentran en las excepciones generadas por la solicitud `ImportTable`.

Errores de validación de datos

Los errores de validación de datos pueden producirse en el nivel de elemento o en el de archivo. Durante la importación, los elementos se validan en función de las reglas de DynamoDB antes de importarlos a la tabla de destino. Cuando un elemento no supera la validación y no se importa, el trabajo de importación lo omite y continúa con el siguiente. Al final del trabajo, el estado de la importación se establece como FAILED (Error) con un FailureCode, un ItemValidationError y un FailureMessage que indica que algunos de los elementos no han superado las comprobaciones de

validación y no se han importado. Consulte los registros de errores de CloudWatch para obtener más información.

Entre las causas más habituales de los errores de validación de datos se encuentran que los objetos no se puedan procesar, que los objetos estén en un formato incorrecto (la entrada especifica DYNAMODB_JSON pero el objeto no está en DYNAMODB_JSON) y que el esquema no coincida con las claves de la tabla de origen especificadas.

Errores de configuración

Los errores de configuración suelen ser errores de flujo de trabajo debidos a la validación de permisos. El flujo de trabajo de importación comprueba algunos permisos después de aceptar la solicitud. Si hay problemas para llamar a alguna de las dependencias requeridas como Amazon S3 o CloudWatch el proceso marca el estado de la importación como FAILED (Error). En `failureCode` y `failureMessage` se indica el motivo del error. Cuando sea aplicable, el mensaje de error también contiene el identificador de la solicitud que puede utilizar para investigar el motivo del error en CloudTrail.

Los errores de configuración más habituales son tener una URL incorrecta para el bucket de Amazon S3 y no tener permiso para acceder al bucket de Amazon S3, a CloudWatch Logs y a las claves AWS KMS utilizadas para descifrar el objeto de Amazon S3. Para obtener más información, consulte Uso y claves de datos.

Validación de objetos de Amazon S3 de origen

Para validar los objetos de S3 de origen, siga estos pasos.

1. Valide el formato de datos y el tipo de compresión

   - Asegúrese de que todos los objetos de Amazon S3 coincidentes con el prefijo especificado tengan el mismo formato (DYNAMODB_JSON, DYNAMODB_ION, CSV)
   - Asegúrese de que todos los objetos de Amazon S3 coincidentes con el prefijo especificado se comprimen de la misma manera (GZIP, ZSTD, NONE)

   > **Note**
   >
   > No es necesario que los objetos de Amazon S3 tengan la extensión correspondiente (.csv/.json/.ion/.gz/.zstd, etc.), ya que tiene prioridad el formato de entrada especificado en la llamada a ImportTable.

2. Valide que los datos de importación se ajustan al esquema de tabla deseado

   - Asegúrese de que cada elemento de los datos de origen tiene la clave principal. La clave de clasificación es opcional para las importaciones.
   - Asegúrese de que el tipo de atributo asociado a la clave principal y a cualquier clave de clasificación coincida con el tipo de atributo de la tabla y del esquema GSI, tal como se especifica en los parámetros de creación de la tabla

Resolución de problemas

Registros de CloudWatch

Para los trabajos de importación con error, se publican mensajes de error detallados en los registros de CloudWatch. Para acceder a estos registros, primero recupere ImportArn de la salida y describe-import mediante este comando:

```
aws dynamodb describe-import --import-arn arn:aws:dynamodb:us-east-1:ACCOUNT:table/
target-table/import/01658528578619-c4d4e311
}
```

Ejemplo de salida:

```
aws dynamodb describe-import --import-arn "arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658528578619-c4d4e311"
{
    "ImportTableDescription": {
        "ImportArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table/
import/01658528578619-c4d4e311",
        "ImportStatus": "FAILED",
        "TableArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table",
        "TableId": "7b7ecc22-302f-4039-8ea9-8e7c3eb2bcb8",
        "ClientToken": "30f8891c-e478-47f4-af4a-67a5c3b595e3",
        "S3BucketSource": {
            "S3BucketOwner": "ACCOUNT",
            "S3Bucket": "my-import-source",
            "S3KeyPrefix": "import-test"
        },
        "ErrorCount": 1,
        "CloudWatchLogGroupArn": "arn:aws:logs:us-east-1:ACCOUNT:log-group:/aws-
dynamodb/imports:*",
        "InputFormat": "CSV",
```

```
        "InputCompressionType": "NONE",
        "TableCreationParameters": {
            "TableName": "target-table",
            "AttributeDefinitions": [
                {
                    "AttributeName": "pk",
                    "AttributeType": "S"
                }
            ],
            "KeySchema": [
                {
                    "AttributeName": "pk",
                    "KeyType": "HASH"
                }
            ],
            "BillingMode": "PAY_PER_REQUEST"
        },
        "StartTime": 1658528578.619,
        "EndTime": 1658528750.628,
        "ProcessedSizeBytes": 70,
        "ProcessedItemCount": 1,
        "ImportedItemCount": 0,
        "FailureCode": "ItemValidationError",
        "FailureMessage": "Some of the items failed validation checks and were not
 imported. Please check CloudWatch error logs for more details."
    }
}
```

Recupere el grupo de registros y el ID de importación de la respuesta anterior y utilícelos para recuperar los registros de errores. El ID de importación es el último elemento de la ruta del campo `ImportArn`. El nombre del grupo de registros es `/aws-dynamodb/imports`. El nombre del flujo de registro de errores es `import-id/error`. Para este ejemplo, sería `01658528578619-c4d4e311/error`.

Falta la clave pk en el elemento

Si el objeto S3 de origen no contiene la clave principal que se ha proporcionado como parámetro, se producirá un error en la importación. Por ejemplo, cuando se define la clave principal para la importación como el nombre de columna "pk".

```
aws dynamodb import-table –s3-bucket-source S3Bucket=my-import-
source,S3KeyPrefix=import-test.csv \
```

```
            –input-format CSV --table-creation-parameters '{"TableName":"target-
table","KeySchema":  \
            [{"AttributeName":"pk","KeyType":"HASH"}],"AttributeDefinitions":
[{"AttributeName":"pk","AttributeType":"S"}],"BillingMode":"PAY_PER_REQUEST"}'
```

Falta la columna "pk" en el objeto de origen `import-test.csv` que tiene el siguiente contenido:

```
title,artist,year_of_release
The Dark Side of the Moon,Pink Floyd,1973
```

Se producirá un error en la importación debido a la falta de la clave principal en el origen de datos.

Registro de errores de CloudWatch de ejemplo:

```
aws logs get-log-events –log-group-name /aws-dynamodb/imports –log-stream-name
 01658528578619-c4d4e311/error
{
"events": [
{
"timestamp": 1658528745319,
"message": "{\"itemS3Pointer\":{\"bucket\":\"my-import-source\",\"key\":
\"import-test.csv\",\"itemIndex\":0},\"importArn\":\"arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658528578619-c4d4e311\",\"errorMessages
\":[\"One or more parameter values were invalid: Missing the key pk in the item\"]}",
"ingestionTime": 1658528745414
}
],
"nextForwardToken": "f/36986426953797707963335499204463414460239026137054642176/s",
"nextBackwardToken": "b/36986426953797707963335499204463414460239026137054642176/s"
}
```

Este registro de errores indica que uno o más valores de los parámetros no eran válidos: falta la clave pk en el elemento. Como se ha producido un error en este trabajo de importación, la tabla "target-table" existe ahora y está vacía porque no se ha importado ningún elemento. Se ha procesado el primer elemento y el objeto no ha superado la validación de elementos.

Para solucionar el problema, elimine primero "target-table" si ya no es necesaria. A continuación, utilice un nombre de columna de clave principal que exista en el objeto de origen o actualice los datos de origen a:

```
pk,title,artist,year_of_release
```

```
Albums::Rock::Classic::1973::AlbumId::ALB25,The Dark Side of the Moon,Pink Floyd,1973
```

La tabla de destino existe

Cuando inicie un trabajo de importación y reciba una respuesta como la siguiente:

```
An error occurred (ResourceInUseException) when calling the ImportTable operation:
 Table already exists: target-table
```

Para solucionar este error, tendrá que elegir un nombre de tabla que no exista ya y volver a intentar la importación.

El bucket especificado no existe

Si el bucket de origen no existe, se producirá un error en la importación y se registrarán los detalles del mensaje de error en CloudWatch.

Importación de descripción de ejemplo:

```
aws dynamodb –endpoint-url $ENDPOINT describe-import –import-arn "arn:aws:dynamodb:us-
east-1:531234567890:table/target-table/import/01658530687105-e6035287"
{
"ImportTableDescription": {
"ImportArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table/
import/01658530687105-e6035287",
"ImportStatus": "FAILED",
"TableArn": "arn:aws:dynamodb:us-east-1:ACCOUNT:table/target-table",
"TableId": "e1215a82-b8d1-45a8-b2e2-14b9dd8eb99c",
"ClientToken": "3048e16a-069b-47a6-9dfb-9c259fd2fb6f",
"S3BucketSource": {
"S3BucketOwner": "531234567890",
"S3Bucket": "BUCKET_DOES_NOT_EXIST",
"S3KeyPrefix": "import-test"
},
"ErrorCount": 0,
"CloudWatchLogGroupArn": "arn:aws:logs:us-east-1:ACCOUNT:log-group:/aws-dynamodb/
imports:*",
"InputFormat": "CSV",
"InputCompressionType": "NONE",
"TableCreationParameters": {
"TableName": "target-table",
"AttributeDefinitions": [
{
```

```
"AttributeName": "pk",
"AttributeType": "S"
}
],
"KeySchema": [
{
"AttributeName": "pk",
"KeyType": "HASH"
}
],
"BillingMode": "PAY_PER_REQUEST"
},
"StartTime": 1658530687.105,
"EndTime": 1658530701.873,
"ProcessedSizeBytes": 0,
"ProcessedItemCount": 0,
"ImportedItemCount": 0,
"FailureCode": "S3NoSuchBucket",
"FailureMessage": "The specified bucket does not exist (Service: Amazon S3; Status
 Code: 404; Error Code: NoSuchBucket; Request ID: Q4W6QYYFDWY6WAKH; S3 Extended Request
 ID: ObqSlLeIMJpQqHLRX2C5Sy7n+8g6iGPwy7ixg7eEeTuEkg/+chU/JF+RbliWytMlkUlUcuCLTrI=;
 Proxy: null)"
}
}
```


`FailureCode` es `S3NoSuchBucket`, con `FailureMessag` que contiene detalles como el ID de la solicitud y el servicio que ha generado el error. Dado que el error se ha detectado antes de importar los datos a la tabla, no se crea una nueva tabla de DynamoDB. En algunos casos, cuando estos errores se producen una vez iniciada la importación de datos, se retiene la tabla con datos parcialmente importados.

Para solucionar este error, asegúrese de que el bucket de Amazon S3 de origen existe y, a continuación, reinicie el proceso de importación.

Prácticas recomendadas para importar de Amazon S3 a DynamoDB

A continuación se describen las prácticas recomendadas para importar datos de Amazon S3 a DynamoDB.

No superar el límite de 50 000 objetos de S3

Cada trabajo de importación admite un máximo de 50 000 objetos de S3. Si su conjunto de datos contiene más de 50 000 objetos, considere la posibilidad de consolidarlos en objetos más grandes.

Evitar objetos S3 excesivamente grandes

Los objetos S3 se importan en paralelo. Disponer de numerosos objetos S3 de tamaño medio permite una ejecución paralela sin una sobrecarga excesiva. En el caso de elementos de menos de 1 KB, considere la posibilidad de colocar 4 000 000 de elementos en cada objeto S3. Si el tamaño promedio de sus elementos es mayor, coloque proporcionalmente menos elementos en cada objeto S3.

Aleatorizar los datos ordenados

Si un objeto S3 contiene los datos ordenados, puede crear una partición activa continua. Se trata de una situación en la que una partición recibe toda la actividad, después la siguiente partición tras esa y así sucesivamente. Los datos ordenados se definen como elementos en secuencia en el objeto S3 que se escribirán en la misma partición de destino durante la importación. Una situación habitual en la que los datos están ordenados es un archivo CSV en el que los elementos están ordenados por clave de partición, de modo que los elementos repetidos comparten la misma clave de partición.

Para evitar una partición activa continua, le recomendamos que aleatorice el orden en estos casos. De este modo, se puede mejorar el rendimiento al distribuir las operaciones de escritura. Para obtener más información, consulte Distribución de la actividad de escritura de forma eficiente al cargar los datos en DynamoDB.

Comprimir datos para mantener el tamaño total del objeto S3 por debajo del límite regional

En el proceso de importación desde S3, existe un límite en el tamaño total de la suma de los datos de objetos S3 que se van a importar. El límite es de 15 TB en las regiones us-east-1, us-west-2 y eu-west-1, y de 1 TB en el resto de las regiones. El límite se basa en los tamaños de los objetos S3 sin procesar.

La compresión permite que quepan más datos brutos según el límite. Si la compresión por sí sola no es suficiente para ajustar la importación según el límite, también puede ponerse en contacto con AWS Premium Support para solicitar un aumento de la cuota.

Tener en cuenta cómo el tamaño del artículo afecta al rendimiento

Si el tamaño de elemento promedio es muy pequeño (inferior a 200 bytes), el proceso de importación puede tardar un poco más que en el caso de elementos de mayor tamaño.

Importar sin ningún índice secundario global

La duración de una tarea de importación puede depender de la presencia de uno o varios índices secundarios globales (GSI). Si tiene previsto establecer índices con claves de partición que tengan

una cardinalidad baja, es posible que la importación sea más rápida si aplaza la creación de índices hasta que finalice la tarea de importación (en lugar de incluirlos en el trabajo de importación).

> **Note**
>
> La creación de un GSI durante la importación no genera gastos de escritura (la creación de un GSI después de la importación sí los generaría).

Exportación de datos de DynamoDB a Amazon S3: cómo funciona

La exportación de DynamoDB a S3 es una solución completamente administrada para exportar sus datos de DynamoDB a un bucket de S3 de Amazon a escala. Mediante la exportación de DynamoDB a S3, puede exportar datos de una tabla de Amazon DynamoDB en cualquier momento dado dentro de su periodo de recuperación en un momento dado (PITR) a un bucket de S3 de Amazon. Debe activar PITR en su tabla para utilizar la función de exportación. Esta característica le permite realizar análisis y consultas complejas de los datos con otros servicios de AWS como Athena, AWS Glue, Amazon SageMaker AI, Amazon EMR y AWS Lake Formation.

La exportación de DynamoDB a S3 le permite exportar datos completos e incrementales de su tabla de DynamoDB. Las exportaciones son asíncronas, no consumen unidades de capacidad de lectura (RCU) y no tienen ningún impacto en el rendimiento y la disponibilidad de las tablas. Los formatos de archivo de exportación compatibles son los formatos JSON de DynamoDB y Amazon Ion. También puede exportar datos a un bucket de S3 propiedad de otra cuenta de AWS y a una región de AWS diferente. Los datos siempre se cifran en ambos extremos.

Las exportaciones completas de DynamoDB se cobran en función del tamaño de la tabla de DynamoDB (datos de la tabla e índices secundarios locales) en el momento en que se realiza la exportación. Las exportaciones incrementales de DynamoDB se cobran en función del tamaño de los datos procesados de sus copias de seguridad continuas correspondientes al periodo de tiempo que se exporta. La exportación incremental tiene un costo mínimo de 10 MB. Se aplican cargos adicionales por el almacenamiento de los datos exportados en Amazon S3 y por las solicitudes de PUT realizadas en su bucket de S3. Para obtener más información sobre estos cargos, consulte los precios de Amazon DynamoDB y los precios de Amazon S3.

Para obtener información específica sobre las cuotas de servicio, consulte Exportar tablas a Amazon S3.

Temas

- Solicitar una exportación de tabla en DynamoDB
- Formato de salida de exportación de tabla de DynamoDB

Solicitar una exportación de tabla en DynamoDB

La exportación de una tabla de DynamoDB le permite exportar datos de la tabla a un bucket de Amazon S3, habilitándolo para realizar análisis y consultas complejas sobre los datos mediante otros servicios de AWS como Athena, AWS Glue, Amazon SageMaker AI, Amazon EMR y AWS Lake Formation. Puede solicitar la exportación de una tabla de DynamoDB mediante la Consola de administración de AWS, la AWS CLI o la API de DynamoDB.

> **Note**
>
> No se admite el pago de buckets de Amazon S3 por parte del solicitante.

DynamoDB admite tanto la exportación completa como la exportación incremental:

- Con las exportaciones completas, puede exportar una instantánea completa de su tabla desde cualquier momento dado dentro del periodo de recuperación en un momento dado (PITR) a su bucket de S3 de Amazon.
- Con las exportaciones incrementales, puede exportar datos de su tabla de DynamoDB que se hayan modificado, actualizado o eliminado entre un periodo de tiempo especificado, dentro de su ventana PITR, a su bucket de S3 de Amazon.

Temas

- Requisitos previos
- Solicitar una exportación mediante la Consola de administración de AWS
- Obtención de detalles sobre exportaciones pasadas en la Consola de administración de AWS
- Solicitar una exportación mediante la AWS CLI
- Obtención de detalles sobre exportaciones pasadas en la AWS CLI
- Solicitud de exportación mediante el SDK de AWS
- Obtención de detalles sobre exportaciones pasadas mediante el SDK de AWS

Requisitos previos

Habilitar PITR

Para utilizar la característica de exportación a S3, debe activar PITR en su tabla. Para más detalles sobre cómo activar PITR, consulte Recuperación en un momento dado. Si solicita una exportación para una tabla que no tiene PITR activado, su solicitud no se realizará correctamente y aparecerá el mensaje de excepción: “An error occurred (PointInTimeRecoveryUnavailableException) when calling the `ExportTableToPointInTime` operation: Point in time recovery is not enabled for table 'my-dynamodb-table’”. Solo puede solicitar y exportar desde un momento determinado que se encuentre dentro de los `RecoveryPeriodInDays` de PITR configurados.

Configuración de permisos de S

Puede exportar los datos de la tabla a cualquier bucket de Amazon S3 en el que tenga permiso para escribir. El bucket de destino no tiene por qué estar en la misma región de AWS o tener el mismo propietario que la tabla de origen. Su política de AWS Identity and Access Management (IAM) debe permitirle realizar acciones S3 (`s3:AbortMultipartUpload`, `s3:PutObject` y`s3:PutObjectAcl`) y la acción de exportación DynamoDB (`dynamodb:ExportTableToPointInTime`). He aquí un ejemplo de una política de muestra que concederá a su usuario permisos para realizar exportaciones a un bucket de S3.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDynamoDBExportAction",
            "Effect": "Allow",
            "Action": "dynamodb:ExportTableToPointInTime",
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table"
        },
        {
            "Sid": "AllowS3BucketWrites",
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
```

```
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

Si necesita escribir en un bucket de Amazon S3 que esté en otra cuenta o en el que no tenga permisos de escritura, el propietario del bucket de Amazon S3 debe agregar una política de buckets para permitirle exportar desde DynamoDB a ese bucket. He aquí un ejemplo de política en el bucket de Amazon S3 de destino.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ExampleStatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:user/Dave"
            },
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

Revocar estos permisos mientras se lleva a cabo una exportación dará como resultado archivos parciales.

> **Note**
>
> Si la tabla o el bucket al que realiza la exportación está cifrado con claves administradas por el cliente, las políticas de esa clave KMS deben dar permiso a DynamoDB para que la utilice. Este permiso se concede a través del usuario o rol de IAM que desencadena el

trabajo de exportación. Para obtener más información sobre el cifrado, incluidas las prácticas recomendadas, consulte ¿Cómo Amazon DynamoDB utiliza AWS KMS? y Using a custom KMS key (Cómo utilizar una clave KMS personalizada).

Solicitar una exportación mediante la Consola de administración de AWS

En el siguiente ejemplo, se muestra cómo utilizar la consola de DynamoDB para exportar una tabla llamada `MusicCollection`.

> **Note**
>
> En este procedimiento se presupone que ha habilitado la recuperación a un momento dado. Para habilitar esta función en la tabla `MusicCollection`, dentro de la tabla, diríjase a la pestaña Overview (Información general) de la sección Table details (Detalles de la tabla) y seleccione la opción Enable (Habilitar) en Point-in-time recovery (Recuperación a un momento dado).

Para solicitar una exportación de tabla

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/.
2. En el panel de navegación del lado izquierdo de la consola, elija Exports to S3 (Exportación a S3).
3. Seleccione el botón Exportar a S3.
4. Elija una tabla de origen y un bucket de S3 de destino. Si el bucket de destino es propiedad de la cuenta, puede utilizar el botón Browse S3 (Examinar S3) para encontrarlo. En caso contrario, ingrese la URL del bucket con el comando `s3://bucketname/prefix format.`. El **`prefix`** es una carpeta opcional para ayudarle a mantener el bucket de destino organizado.
5. Elija Exportación completa o Exportación incremental. Una exportación completa genera la instantánea completa de la tabla tal y como estaba en el momento que especifique. Una exportación incremental da salida a los cambios realizados en su tabla durante el periodo de exportación especificado. Su salida está compactada de tal manera que solo contiene el estado final del elemento del periodo de exportación. El elemento solo aparecerá una vez en la exportación aunque tenga varias actualizaciones en el mismo periodo de exportación.

Full export

1. Seleccione el momento a partir del cual desea exportar la instantánea de la tabla completa. Puede ser en cualquier momento dentro del periodo de PITR. También puede seleccionar Hora actual para exportar la última instantánea.
2. Para Formato de archivo exportado, elija entre DynamoDB JSON y Amazon Ion. De forma predeterminada, la tabla se exportará en formato DynamoDB JSON a partir del último tiempo restaurable en la ventana de recuperación a un momento dado y se cifrará con una clave de Amazon S3 (SSE-S3). Puede modificar esta configuración de exportación si es necesario.

> **Note**
>
> Si elige cifrar la exportación usando una clave protegida por AWS Key Management Service (AWS KMS), la clave debe estar en la misma región que el bucket de S3 de destino.

Incremental export

1. Seleccione el periodo de exportación para el que desea exportar los datos incrementales. Seleccione una hora de inicio en el periodo de PITR. La duración del periodo de exportación debe ser de al menos 15 minutos y no debe superar las 24 horas. La hora de inicio del periodo de exportación es inclusiva y la hora de finalización es exclusiva.
2. Elija entre Modo absoluto o Modo relativo.
   a. Modo absoluto exportará datos incrementales para el periodo de tiempo que especifique.
   b. Modo relativo exportará los datos incrementales durante un período de exportación relativo al tiempo de envío del trabajo de exportación.
3. Para Formato de archivo exportado, elija entre DynamoDB JSON y Amazon Ion. De forma predeterminada, la tabla se exportará en formato DynamoDB JSON a partir del último tiempo restaurable en la ventana de recuperación a un momento dado y se cifrará con una clave de Amazon S3 (SSE-S3). Puede modificar esta configuración de exportación si es necesario.

> **Note**
>
> Si elige cifrar la exportación usando una clave protegida por AWS Key Management Service (AWS KMS), la clave debe estar en la misma región que el bucket de S3 de destino.

4. Para Tipo de vista de exportación, seleccione Imágenes nuevas y antiguas o Solo imágenes nuevas. La nueva imagen proporciona el último estado del elemento. La imagen antigua proporciona el estado del elemento justo antes de la “fecha y hora de inicio” especificadas. La configuración predeterminada es Imágenes nuevas y antiguas. Para obtener más información sobre imágenes nuevas e imágenes antiguas, consulte Salida de exportación incremental.

6. Seleccione Exportar para empezar.

Los datos exportados no son coherentes con las transacciones. Sus operaciones de transacción pueden dividirse en dos salidas de exportación. Una operación de transacción reflejada en la exportación puede modificar un subconjunto de elementos, mientras que otro subconjunto de modificaciones de la misma transacción no se refleja en la misma solicitud de exportación. No obstante, las exportaciones son finalmente coherentes. Si se interrumpe una transacción durante una exportación, tendrá la transacción restante en su siguiente exportación contigua, sin duplicados. Los periodos de tiempo utilizados para las exportaciones se basan en un reloj interno del sistema y pueden variar en un minuto del reloj local de su aplicación.

Obtención de detalles sobre exportaciones pasadas en la Consola de administración de AWS

Puede encontrar información sobre las tareas de exportación que ha ejecutado en el pasado en la sección Exportaciones a S3 de la barra lateral de navegación. Esta sección contiene una lista de todas las exportaciones que ha creado en los últimos 90 días. Seleccione el ARN de una tarea enumerada en la pestaña Exportaciones para recuperar información sobre esa exportación, incluida la configuración avanzada que haya elegido. Tenga en cuenta que aunque los metadatos de la tarea de exportación vencen tras 90 días y los trabajos anteriores a eso ya no se encuentran en esta lista, los objetos de su bucket de S3 permanecen siempre que lo permitan las políticas de bucket. DynamoDB nunca elimina ninguno de los objetos que crea en su bucket de S3 durante una exportación.

Solicitar una exportación mediante la AWS CLI

En el siguiente ejemplo se muestra cómo utilizar la AWS CLI para exportar una tabla existente denominada `MusicCollection` a un bucket de S3 denominado `ddb-export-musiccollection`.

> **Note**
>
> En este procedimiento se presupone que ha habilitado la recuperación a un momento dado. Para habilitarlo en la tabla `MusicCollection`, ejecute el siguiente comando.
>
> ```
> aws dynamodb update-continuous-backups \
>     --table-name MusicCollection \
>     --point-in-time-recovery-specification PointInTimeRecoveryEnabled=True
> ```

Full export

El siguiente comando exporta la `MusicCollection` a un bucket de S3 denominado `ddb-export-musiccollection-9012345678` con un prefijo de `2020-Nov`. Los datos de la tabla se exportarán en formato DynamoDB JSON de un tiempo específico dentro la ventana de recuperación a un momento dado y se cifrará con una clave de Amazon S3 (SSE-S3).

> **Note**
>
> Si solicita una exportación de tablas entre cuentas, asegúrese de incluir la opción `--s3-bucket-owner`.

```
aws dynamodb export-table-to-point-in-time \
  --table-arn arn:aws:dynamodb:us-west-2:123456789012:table/MusicCollection \
  --s3-bucket ddb-export-musiccollection-9012345678 \
  --s3-prefix 2020-Nov \
  --export-format DYNAMODB_JSON \
  --export-time 1604632434 \
  --s3-bucket-owner 9012345678 \
  --s3-sse-algorithm AES256
```

Incremental export

El siguiente comando realiza una exportación incremental proporcionando un archivo nuevo `--export-type` y `--incremental-export-specification`. Sustituya lo que aparece en cursiva por sus propios valores. Los tiempos se especifican en segundos desde el tiempo Unix.

```
aws dynamodb export-table-to-point-in-time \
  --table-arn arn:aws:dynamodb:REGION:ACCOUNT:table/TABLENAME \
  --s3-bucket BUCKET --s3-prefix PREFIX \
  --incremental-export-specification
 ExportFromTime=1693569600,ExportToTime=1693656000,ExportViewType=NEW_AND_OLD_IMAGES
 \
  --export-type INCREMENTAL_EXPORT
```

> **Note**
>
> Si elige cifrar la exportación usando una clave protegida por AWS Key Management Service (AWS KMS), la clave debe estar en la misma región que el bucket de S3 de destino.

Obtención de detalles sobre exportaciones pasadas en la AWS CLI

Puede encontrar información acerca de las solicitudes de exportación que ha ejecutado en el pasado mediante el comando `list-exports`. Este comando devuelve una lista de todas las exportaciones que ha creado en los últimos 90 días. Tenga en cuenta que aunque los metadatos de la tarea de exportación vencen después de 90 días y los trabajos anteriores a eso ya no se regresan por medio del comando `list-exports`, los objetos de su bucket de S3 permanecen siempre que lo permitan las políticas de bucket. DynamoDB nunca elimina ninguno de los objetos que crea en su bucket de S3 durante una exportación.

Las exportaciones tienen un estado de `PENDING` hasta que se realizan o no correctamente. Si se realizan correctamente, el estado cambia a `COMPLETED`. Si no se realizan correctamente, el estado cambia a `FAILED` con un `failure_message` y una `failure_reason`.

En el siguiente ejemplo, utilizamos el parámetro opcional `table-arn` para enumerar solo las exportaciones de una tabla específica.

```
aws dynamodb list-exports \
```

```
    --table-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog
```

Para recuperar información detallada sobre una tarea de exportación específica, incluyendo la configuración avanzada, utilice el comando `describe-export`.

```
aws dynamodb describe-export \
    --export-arn arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/
export/01234567890123-a1b2c3d4
```

Solicitud de exportación mediante el SDK de AWS

Utilice estos fragmentos de código para solicitar una exportación de tablas mediante el SDK de AWS de su elección.

Python

Exportación completa

```
import boto3
from datetime import datetime

# remove endpoint_url for real use
client = boto3.client('dynamodb')

# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/export_table_to_point_in_time.html
client.export_table_to_point_in_time(
    TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
    ExportTime=datetime(2023, 9, 20, 12, 0, 0),
    S3Bucket='bucket',
    S3Prefix='prefix',
    S3SseAlgorithm='AES256',
    ExportFormat='DYNAMODB_JSON'
)
```

Exportación incremental

```
import boto3
from datetime import datetime

client = boto3.client('dynamodb')
```

```
client.export_table_to_point_in_time(
    TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
    IncrementalExportSpecification={
      'ExportFromTime': datetime(2023, 9, 20, 12, 0, 0),
      'ExportToTime': datetime(2023, 9, 20, 13, 0, 0),
      'ExportViewType': 'NEW_AND_OLD_IMAGES'
    },
    ExportType='INCREMENTAL_EXPORT',
    S3Bucket='bucket',
    S3Prefix='prefix',
    S3SseAlgorithm='AES256',
    ExportFormat='DYNAMODB_JSON'
)
```

Obtención de detalles sobre exportaciones pasadas mediante el SDK de AWS

Utilice estos fragmentos de código para obtener detalles sobre las exportaciones de tablas anteriores mediante el SDK de AWS de su elección.

Python

Enumeración

```
import boto3

client = boto3.client('dynamodb')

# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/list_exports.html

print(
  client.list_exports(
     TableArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE',
  )
)
```

Describir

```
import boto3

client = boto3.client('dynamodb')
```

```
# https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/
dynamodb/client/describe_export.html

print(
  client.describe_export(
     ExportArn='arn:aws:dynamodb:us-east-1:0123456789:table/TABLE/
export/01695353076000-06e2188f',
  )['ExportDescription']
)
```

Formato de salida de exportación de tabla de DynamoDB

Una exportación de tabla de DynamoDB incluye archivos de manifiesto además de los archivos que contienen los datos de su tabla. Todos estos archivos se guardan en el bucket de Amazon S3 que especifique en su solicitud de exportación. En las siguientes secciones se describe el formato y el contenido de cada objeto de salida.

Salida de exportación completa

Archivos de manifiesto

DynamoDB crea archivos de manifiesto, junto con sus archivos de suma de comprobación, en el bucket de S3 especificado para cada solicitud de exportación.

```
export-prefix/AWSDynamoDB/ExportId/manifest-summary.json
export-prefix/AWSDynamoDB/ExportId/manifest-summary.checksum
export-prefix/AWSDynamoDB/ExportId/manifest-files.json
export-prefix/AWSDynamoDB/ExportId/manifest-files.checksum
```

Usted elige un **`export-prefix`** cuando solicita una exportación de tabla. Esto le ayuda a mantener organizados los archivos del bucket de S3 de destino. **`ExportId`** es un token único generado por el servicio para garantizar que varias exportaciones al mismo bucket de S3 y `export-prefix` no se sobrescriban entre sí.

La exportación crea al menos un archivo por partición. En el caso de las particiones que estén vacías, su solicitud de exportación creará un archivo vacío. Todos los elementos de cada archivo provienen del espacio de claves hash de esa partición en particular.

> **Note**
>
> DynamoDB también crea un archivo vacío denominado `_started` en el mismo directorio que los archivos de manifiesto. Este archivo comprueba que en el bucket de destino se puede escribir y que la exportación ha comenzado. Se puede eliminar de forma segura.

El manifiesto resumido

El archivo `manifest-summary.json` contiene información de resumen sobre el trabajo de exportación. Esto le permite saber qué archivos de datos de la carpeta de datos compartidos están asociados a esta exportación. Su formato es el siguiente:

```
{
   "version": "2020-06-30",
   "exportArn": "arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog/
export/01234567890123-a1b2c3d4",
   "startTime": "2020-11-04T07:28:34.028Z",
   "endTime": "2020-11-04T07:33:43.897Z",
   "tableArn": "arn:aws:dynamodb:us-east-1:123456789012:table/ProductCatalog",
   "tableId": "12345a12-abcd-123a-ab12-1234abc12345",
   "exportTime": "2020-11-04T07:28:34.028Z",
   "s3Bucket": "ddb-productcatalog-export",
   "s3Prefix": "2020-Nov",
   "s3SseAlgorithm": "AES256",
   "s3SseKmsKeyId": null,
   "manifestFilesS3Key": "AWSDynamoDB/01693685827463-2d8752fd/manifest-files.json",
   "billedSizeBytes": 0,
   "itemCount": 8,
   "outputFormat": "DYNAMODB_JSON",
   "exportType": "FULL_EXPORT"
}
```

El manifiesto de archivos

El archivo `manifest-files.json` contiene información sobre los archivos que contienen los datos de la tabla exportados. El archivo está en formato Líneas JSON, por lo que las líneas nuevas se utilizan como delimitadores de elementos. En el siguiente ejemplo, los detalles de un archivo de datos de un manifiesto de archivos tienen formato en varias líneas para facilitar la legibilidad.

```
{
```

```
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/01693685827463-2d8752fd/data/asdl123dasas.json.gz"
}
```

Archivos de datos

DynamoDB puede exportar los datos de la tabla en dos formatos: DynamoDB JSON y Amazon Ion. Independientemente del formato que elija, sus datos se escribirán en varios archivos comprimidos con los nombres de las claves. Estos archivos también aparecen en el archivo `manifest-files.json`.

La estructura de directorios de su bucket de Amazon S3 después de una exportación completa contendrá todos sus archivos de manifiesto y archivos de datos bajo la carpeta ID de exportación.

```
amzn-s3-demo-bucket/DestinationPrefix
.
### AWSDynamoDB
    ### 01693685827463-2d8752fd     // the single full export
    #   ### manifest-files.json     // manifest points to files under 'data' subfolder
    #   ### manifest-files.checksum
    #   ### manifest-summary.json   // stores metadata about request
    #   ### manifest-summary.md5
    #   ### data                    // The data exported by full export
    #   #   ### asdl123dasas.json.gz
    #   #   ...
    #   ### _started                // empty file for permission check
```

DynamoDB JSON

Una exportación de tabla en formato DynamoDB JSON consta de varios objetos `Item`. Cada objeto individual está en el formato JSON dirigido estándar de DynamoDB.

Al crear analizadores personalizados para los datos de exportación de DynamoDB JSON, el formato es Líneas JSON. Esto significa que las líneas nuevas se utilizan como delimitadores de elementos. Muchos servicios de AWS, como Athena y AWS Glue, analizarán este formato automáticamente.

En el siguiente ejemplo, se ha dado formato a un único elemento de una exportación de DynamoDB JSON en varias líneas para facilitar la legibilidad.

```
{
```

```
    "Item":{
        "Authors":{
            "SS":[
                "Author1",
                "Author2"
            ]
        },
        "Dimensions":{
            "S":"8.5 x 11.0 x 1.5"
        },
        "ISBN":{
            "S":"333-3333333333"
        },
        "Id":{
            "N":"103"
        },
        "InPublication":{
            "BOOL":false
        },
        "PageCount":{
            "N":"600"
        },
        "Price":{
            "N":"2000"
        },
        "ProductCategory":{
            "S":"Book"
        },
        "Title":{
            "S":"Book 103 Title"
        }
    }
}
```

Amazon Ion

Amazon Ion es un formato de serialización de datos jerárquico, autodescriptivo y altamente digitado, creado para abordar los desafíos rápidos de desarrollo, desacoplamiento y eficiencia que surgen todos los días mientras se diseñan arquitecturas a gran escala orientadas a servicios. DynamoDB admite la exportación de datos de tabla en formato de texto de Ion, que es un superconjunto de JSON.

Al exportar una tabla al formato Ion, los tipos de datos de DynamoDB utilizados en la tabla se asignan a los tipos de datos Ion. Los conjuntos de DynamoDB utilizan anotaciones de tipo Ion para desambiguar el tipo de datos utilizado en la tabla de origen.

En la siguiente tabla se muestra la asignación de los tipos de datos de DynamoDB a los tipos de datos de iones:

Tipo de dato de DynamoDB	Representación Ion
Cadena (S)	cadena
Booleano (BOOL)	bool
Número (N)	decimal
Binario (B)	blob
Conjunto (SS, NS, BS)	lista (con anotación de tipo $dynamodb_SS, $dynamodb_NS o $dynamodb_BS)
Enumeración	list
Asignación	struct

Los elementos de una exportación de Ion están delimitados por nuevas líneas. Cada línea comienza con un marcador de versión Ion, seguido de un elemento en formato Ion. En el siguiente ejemplo, se ha dado formato a un elemento de una exportación de Ion en varias líneas para facilitar la legibilidad.

```
$ion_1_0 {
    Item:{
        Authors:$dynamodb_SS::["Author1","Author2"],
        Dimensions:"8.5 x 11.0 x 1.5",
        ISBN:"333-3333333333",
        Id:103.,
        InPublication:false,
        PageCount:6d2,
        Price:2d3,
        ProductCategory:"Book",
        Title:"Book 103 Title"
    }
```

```
}
```

Salida de exportación incremental

Archivos de manifiesto

DynamoDB crea archivos de manifiesto, junto con sus archivos de suma de comprobación, en el bucket de S3 especificado para cada solicitud de exportación.

```
export-prefix/AWSDynamoDB/ExportId/manifest-summary.json
export-prefix/AWSDynamoDB/ExportId/manifest-summary.checksum
export-prefix/AWSDynamoDB/ExportId/manifest-files.json
export-prefix/AWSDynamoDB/ExportId/manifest-files.checksum
```

Usted elige un **`export-prefix`** cuando solicita una exportación de tabla. Esto le ayuda a mantener organizados los archivos del bucket de S3 de destino. **`ExportId`** es un token único generado por el servicio para garantizar que varias exportaciones al mismo bucket de S3 y `export-prefix` no se sobrescriban entre sí.

La exportación crea al menos un archivo por partición. En el caso de las particiones que estén vacías, su solicitud de exportación creará un archivo vacío. Todos los elementos de cada archivo provienen del espacio de claves hash de esa partición en particular.

> **Note**
>
> DynamoDB también crea un archivo vacío denominado `_started` en el mismo directorio que los archivos de manifiesto. Este archivo comprueba que en el bucket de destino se puede escribir y que la exportación ha comenzado. Se puede eliminar de forma segura.

El manifiesto resumido

El archivo `manifest-summary.json` contiene información de resumen sobre el trabajo de exportación. Esto le permite saber qué archivos de datos de la carpeta de datos compartidos están asociados a esta exportación. Su formato es el siguiente:

```
{
 "version": "2023-08-01",
 "exportArn": "arn:aws:dynamodb:us-east-1:599882009758:table/export-test/
export/01695097218000-d6299cbd",
 "startTime": "2023-09-19T04:20:18.000Z",
```

```
 "endTime": "2023-09-19T04:40:24.780Z",
 "tableArn": "arn:aws:dynamodb:us-east-1:599882009758:table/export-test",
 "tableId": "b116b490-6460-4d4a-9a6b-5d360abf4fb3",
 "exportFromTime": "2023-09-18T17:00:00.000Z",
 "exportToTime": "2023-09-19T04:00:00.000Z",
 "s3Bucket": "jason-exports",
 "s3Prefix": "20230919-prefix",
 "s3SseAlgorithm": "AES256",
 "s3SseKmsKeyId": null,
 "manifestFilesS3Key": "20230919-prefix/AWSDynamoDB/01693685934212-ac809da5/manifest-
files.json",
 "billedSizeBytes": 20901239349,
 "itemCount": 169928274,
 "outputFormat": "DYNAMODB_JSON",
 "outputView": "NEW_AND_OLD_IMAGES",
 "exportType": "INCREMENTAL_EXPORT"
}
```

El manifiesto de archivos

El archivo `manifest-files.json` contiene información sobre los archivos que contienen los datos de la tabla exportados. El archivo está en formato Líneas JSON, por lo que las líneas nuevas se utilizan como delimitadores de elementos. En el siguiente ejemplo, los detalles de un archivo de datos de un manifiesto de archivos tienen formato en varias líneas para facilitar la legibilidad.

```
{
"itemCount": 8,
   "md5Checksum": "sQMSpEILNgoQmarvDFonGQ==",
   "etag": "af83d6f217c19b8b0fff8023d8ca4716-1",
   "dataFileS3Key": "AWSDynamoDB/data/sgad6417s6vss4p7owp0471bcq.json.gz"
}
```

Archivos de datos

DynamoDB puede exportar los datos de la tabla en dos formatos: DynamoDB JSON y Amazon Ion. Independientemente del formato que elija, sus datos se escribirán en varios archivos comprimidos con los nombres de las claves. Estos archivos también aparecen en el archivo `manifest-files.json`.

Los archivos de datos para las exportaciones incrementales están todos en una carpeta de datos común en su bucket de S3. Sus archivos de manifiesto se encuentran en su carpeta de ID de exportación.

```
amzn-s3-demo-bucket/DestinationPrefix
.
### AWSDynamoDB
    ### 01693685934212-ac809da5     // an incremental export ID
    #   ### manifest-files.json     // manifest points to files under 'data' folder
    #   ### manifest-files.checksum
    #   ### manifest-summary.json   // stores metadata about request
    #   ### manifest-summary.md5
    #   ### _started                // empty file for permission check
    ### 01693686034521-ac809da5
    #   ### manifest-files.json
    #   ### manifest-files.checksum
    #   ### manifest-summary.json
    #   ### manifest-summary.md5
    #   ### _started
    ### data                        // stores all the data files for incremental
 exports
    #   ### sgad6417s6vss4p7owp0471bcq.json.gz
    #   ...
```

En sus archivos de exportación, la salida de cada elemento incluye una marca temporal que representa cuándo se actualizó ese elemento en su tabla y una estructura de datos que indica si fue una operación `insert`, `update` o `delete`. La marca temporal se basa en un reloj interno del sistema y puede variar con respecto al reloj de su aplicación. En el caso de las exportaciones incrementales, puede elegir entre dos tipos de vistas de exportación para su estructura de salida: imágenes nuevas y antiguas o solo imágenes nuevas.

- La nueva imagen proporciona el último estado del elemento
- La imagen antigua proporciona el estado del elemento justo antes de la fecha y hora de inicio especificadas

Los tipos de vista pueden ser útiles si desea ver cómo se modificó el elemento en el periodo de exportación. También puede ser útil para actualizar eficazmente sus sistemas descendentes, especialmente si dichos sistemas tienen una clave de partición que no es la misma que su clave de partición de DynamoDB.

Puede deducir si un elemento de su salida de exportación incremental era una operación `insert`, `update` o `delete` examinando la estructura de la salida. La estructura de la exportación incremental

y sus operaciones correspondientes se resumen en la tabla siguiente para ambos tipos de vistas de exportación.

Operación	Solo imágenes nuevas	Imágenes nuevas y antiguas
Inserte	Claves + nueva imagen	Claves + nueva imagen
Actualización	Claves + nueva imagen	Claves + imagen nueva + imagen antigua
Delete	Claves	Claves + imagen antigua
Insertar + eliminar	Sin salida	Sin salida

DynamoDB JSON

Una exportación de tabla en formato JSON de DynamoDB consta de una marca temporal de metadatos que indica la hora de escritura del elemento, seguida de las claves del elemento y los valores. En el siguiente ejemplo se muestra una salida JSON de DynamoDB mediante la exportación del tipo de vista como imágenes nuevas y antiguas.

```
// Ex 1: Insert
//   An insert means the item did not exist before the incremental export window
//   and was added during the incremental export window

{
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Keys": {
     "PK": {
       "S": "CUST#100"
     }
   },
   "NewImage": {
     "PK": {
       "S": "CUST#100"
     },
     "FirstName": {
       "S": "John"
     },
```

```
      "LastName": {
        "S": "Don"
      }
    }
}

// Ex 2: Update
//   An update means the item existed before the incremental export window
//   and was updated during the incremental export window.
//   The OldImage would not be present if choosing "New images only".

{
    "Metadata": {
      "WriteTimestampMicros": "1680109764000000"
    },
    "Keys": {
      "PK": {
        "S": "CUST#200"
      }
    },
    "OldImage": {
      "PK": {
        "S": "CUST#200"
      },
      "FirstName": {
        "S": "Mary"
      },
      "LastName": {
        "S": "Grace"
      }
    },
    "NewImage": {
      "PK": {
        "S": "CUST#200"
      },
      "FirstName": {
        "S": "Mary"
      },
      "LastName": {
        "S": "Smith"
      }
    }
}
```

```
// Ex 3: Delete
//   A delete means the item existed before the incremental export window
//   and was deleted during the incremental export window
//   The OldImage would not be present if choosing "New images only".

{
   "Metadata": {
     "WriteTimestampMicros": "1680109764000000"
   },
   "Keys": {
     "PK": {
       "S": "CUST#300"
     }
   },
   "OldImage": {
     "PK": {
       "S": "CUST#300"
     },
     "FirstName": {
       "S": "Jose"
     },
     "LastName": {
       "S": "Hernandez"
     }
   }
}

// Ex 4: Insert + Delete
//   Nothing is exported if an item is inserted and deleted within the
//   incremental export window.
```

Amazon Ion

Amazon Ion es un formato de serialización de datos jerárquico, autodescriptivo y altamente digitado, creado para abordar los desafíos rápidos de desarrollo, desacoplamiento y eficiencia que surgen todos los días mientras se diseñan arquitecturas a gran escala orientadas a servicios. DynamoDB admite la exportación de datos de tabla en formato de texto de Ion, que es un superconjunto de JSON.

Al exportar una tabla al formato Ion, los tipos de datos de DynamoDB utilizados en la tabla se asignan a los tipos de datos Ion. Los conjuntos de DynamoDB utilizan anotaciones de tipo Ion para desambiguar el tipo de datos utilizado en la tabla de origen.

En la siguiente tabla se muestra la asignación de los tipos de datos de DynamoDB a los tipos de datos de iones:

Tipo de dato de DynamoDB	Representación Ion
Cadena (S)	cadena
Booleano (BOOL)	bool
Número (N)	decimal
Binario (B)	blob
Conjunto (SS, NS, BS)	lista (con anotación de tipo $dynamodb_SS, $dynamodb_NS o $dynamodb_BS)
Enumeración	list
Asignación	struct

Los elementos de una exportación de Ion están delimitados por nuevas líneas. Cada línea comienza con un marcador de versión Ion, seguido de un elemento en formato Ion. En el siguiente ejemplo, se ha dado formato a un elemento de una exportación de Ion en varias líneas para facilitar la legibilidad.

```
$ion_1_0 {
    Record:{
        Keys:{
             ISBN:"333-3333333333"
         },
        Metadata:{
            WriteTimestampMicros:1684374845117899.
        },
        OldImage:{
            Authors:$dynamodb_SS::["Author1","Author2"],
            ISBN:"333-3333333333",
            Id:103.,
            InPublication:false,
            ProductCategory:"Book",
            Title:"Book 103 Title"
        },
        NewImage:{
```

```
            Authors:$dynamodb_SS::["Author1","Author2"],
            Dimensions:"8.5 x 11.0 x 1.5",
            ISBN:"333-3333333333",
            Id:103.,
            InPublication:true,
            PageCount:6d2,
            Price:2d3,
            ProductCategory:"Book",
            Title:"Book 103 Title"
        }
    }
}
```

Integración sin ETL de DynamoDB con Amazon SageMaker Lakehouse

La integración sin ETL de DynamoDB con Amazon SageMaker Lakehouse elimina la necesidad de crear canalizaciones de traslado de datos personalizadas al replicar de forma automática los datos de DynamoDB en Amazon SageMaker Lakehouse. Esta integración sin código ayuda a los clientes a ejecutar cargas de trabajo de análisis en los datos de DynamoDB mediante Amazon SageMaker Lakehouse sin consumir ninguna capacidad de tablas de DynamoDB. La integración exporta automáticamente los datos de la tabla y mantiene el destino actualizado, normalmente en un plazo de 15 a 30 minutos.

Temas

- Integración sin ETL de DynamoDB con Amazon SageMaker Lakehouse

Integración sin ETL de DynamoDB con Amazon SageMaker Lakehouse

Configurar una integración entre la tabla de DynamoDB y Amazon SageMaker Lakehouse requiere requisitos previos como la configuración de los roles de IAM, que AWS Glue utiliza para acceder a los datos del origen y escribir en el destino, y el uso de claves de KMS para cifrar los datos en la ubicación intermedia o de destino.

Temas

- Requisitos previos a la creación de una integración sin ETL de DynamoDB con Amazon SageMaker Lakehouse
- Creación de una integración sin ETL de Amazon DynamoDB con Amazon SageMaker Lakehouse

- Visualización de métricas de CloudWatch para la integración

Requisitos previos a la creación de una integración sin ETL de DynamoDB con Amazon SageMaker Lakehouse

Para configurar una integración sin ETL con un origen de DynamoDB, debe establecer una política de acceso basada en recursos (RBAC) que permita a AWS Glue acceder a los datos de la tabla de DynamoDB y exportarlos. La política debe incluir permisos específicos como `ExportTableToPointInTime`, `DescribeTable` y `DescribeExport` con condiciones que restrinjan el acceso a una región y una Cuenta de AWS específicas. Consulte Configuración de un origen de Amazon DynamoDB para obtener más información.

La recuperación en un momento dado (PITR) debe estar habilitada para la tabla, y puede aplicar la política mediante comandos de la AWS CLI. La política se puede afinar aún más si se especifica el ARN de integración completo para lograr un control de acceso más restrictivo. Para obtener más información, consulte Requisitos previos para configurar una integración sin ETL.

Creación de una integración sin ETL de Amazon DynamoDB con Amazon SageMaker Lakehouse

Una vez completados los requisitos previos de la integración, puede crear, modificar o eliminar la integración sin ETL según las siguientes indicaciones:

Creación de una integración

Para crear una integración

1. Inicie sesión en la Consola de administración de AWS y abra la consola de Amazon DynamoDB en https://console.aws.amazon.com/dynamodbv2.
2. En el panel de navegación, elija Integraciones.
3. Seleccione Crear una integración sin ETL con Amazon SageMaker Lakehouse y, a continuación, elija Siguiente.
4. Para crear una integración, consulte Creación de una integración.
5. Para modificar una integración, consulte Modificación de una integración.
6. Para eliminar una integración, consulte Eliminación de una integración.
7. Para configurar una integración entre cuentas, consulte Configuración de una integración entre cuentas.

Habilitación de la compactación en las tablas de Amazon S3 de destino

Puede habilitar la compactación para mejorar el rendimiento de las consultas en Amazon Athena.

En primer lugar, complete la configuración de requisitos previos para los recursos de compactación, incluida la configuración del rol de IAM necesario. Consulte la documentación de Lake Formation para obtener información detallada sobre los pasos de configuración del rol de IAM. Consulte Optimización de tablas para la compactación.

Para habilitar la compactación en la tabla de AWS Glue creada durante la integración, siga el proceso de habilitación de la compactación de Lake Formation. Esto lo ayudará a optimizar el rendimiento de la tabla y la eficacia de las consultas.

Visualización de métricas de CloudWatch para la integración

Una vez que se complete la integración, podrá ver estas métricas de CloudWatch y notificaciones de EventBridge generadas en la cuenta para cada trabajo de AWS Glue. Para obtener más información, consulte Supervisión de una integración.

Integración sin ETL de DynamoDB con Amazon OpenSearch Service

Amazon DynamoDB ofrece una integración sin ETL con Amazon OpenSearch Service mediante el complemento de DynamoDB para OpenSearch Ingestion. Amazon OpenSearch Ingestion ofrece una experiencia completamente administrada y sin código para la ingesta de datos en Amazon OpenSearch Service.

El complemento de DynamoDB para OpenSearch Ingestion permite usar una o más tablas de DynamoDB como origen de la ingesta para uno o más índices de OpenSearch Service. Puede explorar y configurar sus canalizaciones de OpenSearch Ingestion con DynamoDB como origen desde las integraciones de OpenSearch Ingestion o de DynamoDB en la Consola de administración de AWS.

- Siga la OpenSearch Ingestion getting started guide para empezar a usar OpenSearch Ingestion.
- Obtenga información sobre los requisitos previos y todas las opciones de configuración del complemento de DynamoDB en la DynamoDB plugin for OpenSearch Ingestion documentation.

Funcionamiento

El complemento utiliza la exportación de DynamoDB a Amazon S3 para crear una instantánea inicial para cargarla en OpenSearch. Una vez cargada la instantánea, el complemento utiliza DynamoDB Streams para replicar cualquier otro cambio prácticamente en tiempo real. Cada elemento se procesa como un evento en OpenSearch Ingestion y se puede modificar con los complementos del procesador. Puede eliminar atributos o crear atributos compuestos y enviarlos a diferentes índices a través de rutas.

Debe tener activada la recuperación en un momento dado (PITR) para utilizar la exportación a Amazon S3. También debe tener DynamoDB Streams activado (con la opción imágenes nuevas y antiguas seleccionada) para poder usarlo. Para crear una canalización sin realizar instantáneas debe excluir la configuración de exportación.

También se puede crear una canalización con solo una instantánea y sin actualizaciones excluyendo la configuración de las secuencias. El complemento no utiliza el rendimiento de lectura o escritura en la tabla, por lo que se puede usar de forma segura sin que afecte al tráfico de producción. Hay límites para la cantidad de consumidores paralelos en una secuencia que debes tener en cuenta antes de crear esta u otras integraciones. Para otras consideraciones, consulte the section called “Prácticas recomendadas de integración”.

Para las canalizaciones simples, una sola unidad de computación de OpenSearch (OCU) puede procesar aproximadamente 1 MB por segundo de escrituras. Esto equivale a unas 1000 unidades de solicitud de escritura (WCU). Puede superar o quedarse por debajo de esta cantidad en función de la complejidad de la canalización y de otros factores.

OpenSearch Ingestion admite una cola de mensajes fallidos (DLQ) para los eventos que provocan errores irrecuperables. Además, la canalización puede reanudarse desde se dejó sin la intervención del usuario, incluso si se interrumpe el servicio con DynamoDB, la canalización o Amazon OpenSearch Service.

Si la interrupción se prolonga durante más de 24 horas, se pueden perder las actualizaciones. Sin embargo, la canalización seguiría procesando las actualizaciones disponibles todavía cuando se restableciera el servicio. Tendría que crear un nuevo índice para corregir cualquier irregularidad derivada de la interrupción de los eventos, a menos que estuvieran en la cola de mensajes fallidos.

Para conocer todos los ajustes y detalles del complemento, consulte la OpenSearch Ingestion DynamoDB plugin documentation.

Experiencia de creación integrada a través de la consola

DynamoDB y OpenSearch Service ofrecen una experiencia integrada en la Consola de administración de AWS, lo que agiliza el proceso inicial. Al seguir estos pasos, el servicio seleccionará automáticamente el esquema de DynamoDB y añadirá la información de DynamoDB adecuada para usted.

Para crear una integración, consulte la guía de introducción a OpenSearch Ingestion. Cuando llegue al punto Step 3: Create a pipeline, sustituya los pasos 1 y 2 por los siguientes:

1. Navegue hasta la consola de DynamoDB.
2. En el panel de navegación de la izquierda, elija Integration.
3. Seleccione la tabla de DynamoDB que desea replicar en OpenSearch.
4. Seleccione Crear.

A partir de aquí, ya puede continuar con el resto del tutorial.

Siguientes pasos

Para comprender mejor cómo se integra DynamoDB con OpenSearch Service, consulte los siguientes temas:

- Getting started with Amazon OpenSearch Ingestion
- DynamoDB plugin configuration and requirements

Gestión de cambios importantes en el índice

OpenSearch puede añadir nuevos atributos a su índice de forma dinámica. Sin embargo, después de configurar la plantilla de asignación para una clave determinada, tendrá que tomar medidas adicionales para cambiarla. Además, si el cambio requiere que vuelva a procesar todos los datos de la tabla de DynamoDB, tendrá que tomar medidas para iniciar una nueva exportación.

> **Note**
>
> En todas estas opciones, es posible que siga teniendo problemas si la tabla de DynamoDB tiene conflictos de tipo con la plantilla de asignación que ha especificado. Asegúrese de tener activada una cola de mensajes fallidos (DLQ) (incluso en desarrollo). Esto facilita la

comprensión sobre lo que puede estar mal en el registro que provoca un conflicto cuando se indexa en su índice en OpenSearch.

Temas

- Funcionamiento
- Eliminación del índice y restablecimiento de la canalización (opción centrada en la canalización)
- Recreación del índice y restablecimiento de la canalización (opción centrada en el índice)
- Creación de un índice y un receptor nuevos (opción en línea)
- Prácticas recomendadas para evitar y depurar los conflictos de tipos

Funcionamiento

Esta es una visión global rápida de las medidas que se deben tomar al gestionar cambios importantes en un índice. Consulte los procedimientos paso a paso en las secciones siguientes.

- Detener e iniciar la canalización: esta opción restablece el estado de la canalización y esta se reiniciará con una nueva exportación completa. No es destructivo, por lo que no elimina el índice ni ningún dato de DynamoDB. Si no crea un índice nuevo antes de hacerlo, es posible que se produzca un gran número de errores debido a conflictos de versiones, ya que la exportación intenta insertar documentos más antiguos que los actuales de `_version` del índice. Estos errores se pueden ignorar tranquilamente. No se le facturará por la canalización mientras esté detenida.
- Actualizar la canalización: esta opción actualiza la configuración de la canalización con un enfoque azul/verde, sin perder ningún estado. Si realizas cambios importantes en la canalización (como añadir nuevas rutas, índices o claves a los índices existentes), es posible que tengas que restablecer completamente la canalización y volver a crear el índice. Esta opción no realiza una exportación completa.
- Eliminar y volver a crear el índice: esta opción elimina los datos y la configuración de asignación del índice. Debe hacerlo antes de realizar cualquier cambio importante en las asignaciones. Romperá cualquier aplicación que dependa del índice hasta que se vuelva a crear y sincronizar el índice. Al eliminar el índice no se inicia ninguna nueva exportación. Deberías eliminar el índice después de actualizar la canalización. De lo contrario, es posible que se vuelva a crear el índice antes de que se actualice la configuración.

Eliminación del índice y restablecimiento de la canalización (opción centrada en la canalización)

Este método suele ser la opción más rápida si aún se encuentra en la fase de desarrollo. Deberá eliminar el índice en OpenSearch Service y, a continuación, detener e iniciar la canalización para iniciar una nueva exportación de todos los datos. De esta manera se garantiza que no haya conflictos entre las plantillas de asignación y los índices existentes y que no se pierdan datos porque la tabla no se ha procesado por completo.

1. Detenga la canalización desde la Consola de administración de AWS o use la operación de la API StopPipeline con la AWS CLI o un SDK.
2. Actualice la configuración de la canalización con los nuevos cambios.
3. Elimine el índice en OpenSearch Service, ya sea mediante una llamada a la API `REST` o desde el panel de OpenSearch.
4. Inicie la canalización desde la consola o use la operación de la API `StartPipeline` con la AWS CLI o un SDK.

   > **Note**
   >
   > Al hacerlo, se inicia una nueva exportación completa, que incurrirá en costos adicionales.

5. Monitoree cualquier problema inesperado, pues se genera una nueva exportación para crear el nuevo índice.
6. Confirme que el índice coincide con sus expectativas en OpenSearch Service.

Cuando se complete la exportación y se reanude la lectura de la secuencia, los datos de la tabla de DynamoDB estarán disponibles en el índice.

Recreación del índice y restablecimiento de la canalización (opción centrada en el índice)

Este método es adecuado si necesita realizar muchas iteraciones en el diseño del índice en OpenSearch Service antes de reanudar la canalización desde DynamoDB. Esto puede resultar útil para el desarrollo si se quiere iterar rápidamente los patrones de búsqueda y quiere evitar tener que esperar a que se completen nuevas exportaciones entre cada iteración.

1. Detenga la canalización desde la Consola de administración de AWS o llame a la operación de la API StopPipeline con la AWS CLI o un SDK.
2. Elimine y vuelva a crear el índice en OpenSearch con la plantilla de asignación que desee usar. Puede insertar manualmente algunos datos de muestra para confirmar que las búsquedas funcionan según lo previsto. Si los datos de muestra pueden entrar en conflicto con algún dato de DynamoDB, asegúrese de eliminarlos antes de continuar con el siguiente paso.
3. Si tiene una plantilla de indexación en proceso, elimínela o sustitúyala por una que ya haya creado en OpenSearch Service. Asegúrese de que el nombre de su índice coincida con el nombre de la canalización.
4. Inicie la canalización desde la consola o llame a la operación de la API `StartPipeline` con la AWS CLI o un SDK.

   > **Note**
   >
   > Al hacerlo, se inicia una nueva exportación completa, que incurrirá en costos adicionales.

5. Monitoree cualquier problema inesperado, pues se genera una nueva exportación para crear el nuevo índice.

Cuando se complete la exportación y se reanude la lectura de la secuencia, los datos de la tabla de DynamoDB estarán disponibles en el índice.

Creación de un índice y un receptor nuevos (opción en línea)

Este método funciona bien si tiene que actualizar su plantilla de asignación, pero está utilizando el índice en producción. Esto crea un índice completamente nuevo, al que tendrá que mover la aplicación después de sincronizarla y validarla.

> **Note**
>
> De este modo se creará otro consumidor en la secuencia. Esto puede ser un problema si también tiene otros consumidores como AWS Lambda o tablas globales. Es posible que tenga que pausar las actualizaciones de la canalización existente para crear capacidad para cargar el nuevo índice.

1. Cree una nueva canalización con una nueva configuración y un nombre de índice diferente.
2. Monitoree el nuevo índice para detectar cualquier problema inesperado.
3. Cambie la aplicación al nuevo índice.
4. Detenga y elimine la canalización anterior después de comprobar que todo funciona correctamente.

Prácticas recomendadas para evitar y depurar los conflictos de tipos

- Utilice siempre una cola de mensajes fallidos (DLQ) para facilitar la depuración cuando haya conflictos de tipos.
- Utilice siempre una plantilla de índice con asignaciones y establezca `include_keys`. Si bien OpenSearch Service asigna nuevas claves de forma dinámica, esto puede provocar problemas con comportamientos inesperados (como esperar que algo sea un `GeoPoint`, pero se crea como `string` o `object`) o errores (como tener un `number` que es una combinación de valores `long` y `float`).
- Si necesita que el índice actual siga funcionando durante la fase de producción, también puede sustituir cualquiera de los pasos anteriores para eliminar el índice y cambiarle el nombre al índice en el archivo de configuración de la canalización. Al hacerlo se crea un índice completamente nuevo. A continuación, tendrá que actualizar la aplicación para que apunte hacia el nuevo índice una vez que se haya completado.
- Si tiene un problema de conversión de tipos que haya solucionado con un procesador, puede probarlo con `UpdatePipeline`. Para ello, tendrá que detener e iniciar o procesar las colas de mensajes fallidos para corregir cualquier documento que se hubiera omitido previamente y que tuviera errores.

Prácticas recomendadas para trabajar con la integración sin ETL de DynamoDB y OpenSearch Service

DynamoDB dispone de una integración sin ETL de DynamoDB con Amazon OpenSearch Service. Para obtener más información, consulte el complemento de DynamoDB para OpenSearch Ingestion y las prácticas recomendadas específicas para Amazon OpenSearch Service.

Configuración

- Indexe únicamente los datos en los que tenga que realizar búsquedas. Utilice siempre una plantilla de asignación (`template_type: index_template` y `template_content`) y `include_keys` para implementarla.
- Monitoree los registros para detectar errores relacionados con conflictos con los tipos. OpenSearch Service espera que todos los valores de una clave determinada sean del mismo tipo. Si hay alguna discrepancia, genera excepciones. Si se topa con alguno de estos errores, puede añadir un procesador para detectar que una clave determinada tenga siempre el mismo valor.
- De forma general, utilice el valor de los metadatos `primary_key` para el valor `document_id`. En OpenSearch Service, el ID del documento equivale a la clave principal de DynamoDB. Al usar la clave principal es más sencillo buscar el documento y garantizar que las actualizaciones se repliquen en él de forma coherente y sin conflictos.

  Puede usar la función auxiliar `getMetadata` para obtener la clave principal (por ejemplo, `document_id: "${getMetadata('primary_key')}"`). Si utiliza una clave primaria compuesta, la función auxiliar las concatenará automáticamente.
- Se recomienda utilizar el valor de los metadatos `opensearch_action` para la configuración `action`. De este modo se garantiza que las actualizaciones se repliquen de forma que los datos de OpenSearch Service coincidan con el estado más reciente de DynamoDB.

  Puede usar la función auxiliar `getMetadata` para obtener la clave principal (por ejemplo, `action: "${getMetadata('opensearch_action')}"`). También puede usar el tipo de evento de secuencia mediante `dynamodb_event_name` para casos de uso como el filtrado. Sin embargo, no debería usarlo para la configuración `action`.

Observabilidad

- Utilice siempre una cola de mensajes fallidos (DLQ) en los receptores de OpenSearch para gestionar los eventos eliminados. DynamoDB suele estar menos estructurado que OpenSearch Service, por lo que es posible que ocurra algo inesperado. Las colas de mensajes fallidos permiten recuperar eventos individuales e incluso automatizar el proceso de recuperación. De este modo, se evita tener que reconstruir todo el índice.
- Establezca siempre alertas para indicar que el retardo en la replicación no supere un valor esperado. Por lo general, es prudente prever un minuto sin que la alerta sea demasiado

escandalosa. Esto puede variar en función del pico del tráfico de escritura y de la configuración de la unidad de computación de OpenSearch (OCU) en la canalización.

Si el retardo en la replicación es superior a 24 horas, la secuencia empezará a eliminar eventos y tendrá problemas de precisión, a menos que reconstruya completamente el índice desde cero.

Escalado

- Utilice el escalado automático para las canalizaciones a fin de escalar o reducir verticalmente las OCU para que se adapten mejor a la carga de trabajo.
- Para las tablas de rendimiento aprovisionadas sin escalado automático, recomendamos configurar las OCU en función de la cantidad de unidades de capacidad de escritura (WCU) dividida entre 1000. Establezca el valor mínimo en 1 OCU por debajo de esa cantidad (pero al menos 1) y defina el valor máximo en al menos 1 OCU por encima de esa cantidad.
  - Fórmula:

    ```
    OCU_minimum = GREATEST((table_WCU / 1000) - 1, 1)
    OCU_maximum = (table_WCU / 1000) + 1
    ```

  - Ejemplo: La tabla tiene 25 000 WCU aprovisionadas. Las OCU de su canalización deben configurarse con un valor mínimo de 24 (25 000/1000 - 1) y un valor máximo de 26 (25 000/1000 + 1).
- Para las tablas de rendimiento aprovisionadas con escalado automático, recomendamos configurar las OCU en función de las cantidades mínima y máxima de WCU divididas entre 1000. Establezca el valor mínimo en 1 OCU por debajo del valor mínimo de DynamoDB y defina el valor máximo en al menos 1 OCU por encima del valor máximo de DynamoDB.
  - Fórmula:

    ```
    OCU_minimum = GREATEST((table_minimum_WCU / 1000) - 1, 1)
    OCU_maximum = (table_maximum_WCU / 1000) + 1
    ```

  - Ejemplo: La tabla tiene una política de escalado automático con un valor mínimo de 8000 y un valor máximo de 14 000. Las OCU de su canalización deben configurarse con un valor mínimo de 7 (8000/1000 - 1) y un valor máximo de 15 (14 000/1000 + 1).
- Para las tablas de rendimiento bajo demanda, recomendamos configurar las OCU en función del pico y el valle típicos para las unidades de solicitud de escritura por segundo. Es posible que tenga que calcular una media de un período de tiempo más largo, en función de la agregación

disponible. Establezca el valor mínimo en 1 OCU por debajo del valor mínimo de DynamoDB y defina el valor máximo en al menos 1 OCU por encima del valor máximo de DynamoDB.

- Fórmula:

```
# Assuming we have writes aggregated at the minute level
OCU_minimum = GREATEST((min(table_writes_1min) / (60 * 1000)) - 1, 1)
OCU_maximum = (max(table_writes_1min) / (60 * 1000)) + 1
```

- Ejemplo: La tabla tiene un valor medio de 300 unidades de solicitud de escritura por segundo y un pico medio de 4300. Las OCU de su canalización deben configurarse con un valor mínimo de 1 (300/1000 - 1, pero al menos 1) y un valor máximo de 5 (4300/1000 + 1).

- Siga las prácticas recomendadas para escalar los índices de OpenSearch Service de destino. Si los índices no se han escalado suficientemente, se ralentizará la ingesta desde DynamoDB y se podrían generar retardos.

> **Note**
>
> `GREATEST` es una función de SQL que devuelve el argumento con el valor más alto según un conjunto de argumentos determinado.

Integración de DynamoDB con Amazon EventBridge

Amazon DynamoDB ofrece DynamoDB Streams para la captura de datos de cambios, lo que permite capturar los cambios en elementos en las tablas de DynamoDB. DynamoDB Streams puede invocar funciones de Lambda para procesar esos cambios; de esta forma, se puede utilizar una integración basada en eventos con otros servicios y aplicaciones. DynamoDB Streams también admite el filtrado, lo que permite realizar un procesamiento de eventos eficaz y específico.

DynamoDB Streams admite hasta dos consumidores simultáneos por partición y permite realizar el filtrado mediante filtrado de eventos de Lambda para que solo se procesen los elementos que cumplan unos criterios específicos. Puede que algunos clientes necesiten atender a más de dos consumidores. Es posible que otros tengan que enriquecer los eventos de cambio antes de procesarlos o utilizar un filtrado y enrutamiento más avanzados.

La integración de DynamoDB con EventBridge puede satisfacer esos requisitos.

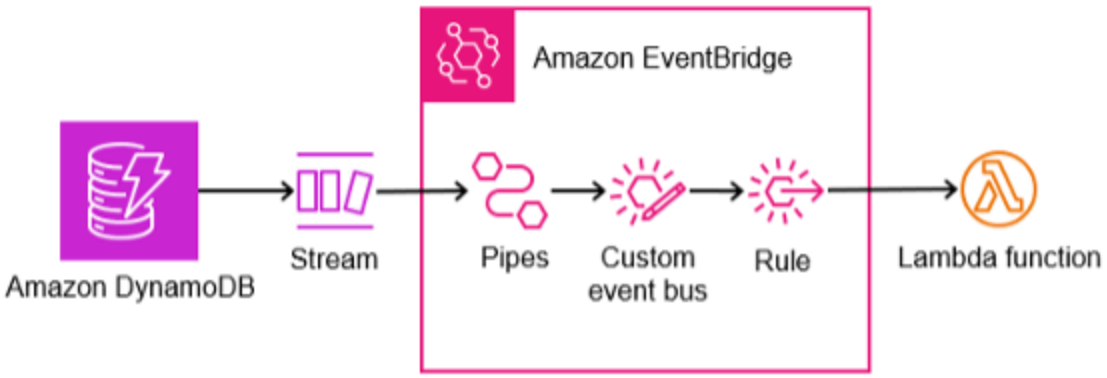
Amazon EventBridge es un servicio sin servidor que utiliza eventos para conectar los componentes de la aplicación entre sí, lo que facilita la creación de aplicaciones escalables basadas en eventos. EventBridge ofrece una integración nativa con Amazon DynamoDB a través de canalizaciones de EventBridge, lo que permite conseguir un flujo de datos sin interrupciones desde DynamoDB a un bus de EventBridge. Luego, ese bus puede distribuirse a múltiples aplicaciones y servicios mediante un conjunto de reglas y objetivos.

Temas

- Funcionamiento
- Creación de una integración a través de la consola
- Siguientes pasos

Funcionamiento

La integración entre DynamoDB y las canalizaciones de EventBridge utiliza DynamoDB Streams para capturar una secuencia ordenada en el tiempo de cambios en elementos en una tabla de DynamoDB. Cada registro que se captura de esta manera contiene los datos modificados de la tabla.



Una canalización de EventBridge consume eventos de DynamoDB Streams y los enruta a un destino, como un bus de EventBridge (un bus de eventos es un enrutador que recibe eventos y los entrega en un punto, que también se denomina destino). La entrega se basa en reglas que coinciden con el contenido del evento. Opcionalmente, la canalización también incluye la capacidad de filtrar eventos específicos y realizar enriquecimientos en los datos de eventos antes de enviarlos al destino.

Aunque EventBridge admite varios tipos de destinos, una opción habitual a la hora de implementar un diseño de distribución es utilizar una función de Lambda como destino. El siguiente ejemplo demuestra una integración con un destino de una función de Lambda.

Creación de una integración a través de la consola

Siga los pasos que se indican a continuación para crear una integración mediante la Consola de administración de AWS.

1. Habilite DynamoDB Streams en la tabla de origen. Para ello, siga los pasos de la sección Habilitación de una secuencia de la Guía para desarrolladores de DynamoDB. Si DynamoDB Streams ya está habilitado en la tabla de origen, compruebe que actualmente haya menos de dos consumidores. Los consumidores pueden ser funciones de Lambda, tablas globales de DynamoDB, integraciones sin ETL de Amazon DynamoDB con Amazon OpenSearch Service o aplicaciones que leen directamente los flujos, por ejemplo, a través del adaptador Kinesis de DynamoDB Streams.
2. Cree un bus de eventos de EventBridge. Para ello, siga los pasos que se indican en la sección Creating an Amazon EventBridge event bus de la Guía del usuario de EventBridge.
   a. Al crear el bus de eventos, habilite Detección de esquemas.
3. Cree una canalización de EventBridge. Para ello, siga los pasos que se indican en la sección Creating an Amazon EventBridge pipe de la Guía del usuario de EventBridge.
   a. Al configurar el origen, en el campo Origen, seleccione DynamoDB y, en el campo Flujos de DynamoDB, elija el nombre del flujo de la tabla de origen.
   b. Al configurar el destino, en el campo Servicio de destino, seleccione Bus de eventos de EventBridge y, en el campo Bus de eventos como destino, seleccione el bus de eventos creado en el paso 2.
4. Escriba un elemento de ejemplo en la tabla de DynamoDB de origen para activar un evento. Esto permitirá a EventBridge deducir el esquema del elemento de ejemplo. Este esquema se puede usar para crear reglas de enrutamiento de eventos. Por ejemplo, si va a implementar un patrón de diseño que implica sobrecargar los atributos, es posible que desee activar reglas diferentes en función del valor de la clave de clasificación. Encontrará información detallada sobre cómo escribir un elemento en DynamoDB en la sección Working with items and attributes de la Guía para desarrolladores de DynamoDB.
5. Cree una función de Lambda de Python de ejemplo para utilizarla como destino. Para ello, siga los pasos de la sección Creación de funciones de Lambda con Python de la Guía para desarrolladores de Lambda. Cuando cree la función, puede utilizar el siguiente código de ejemplo

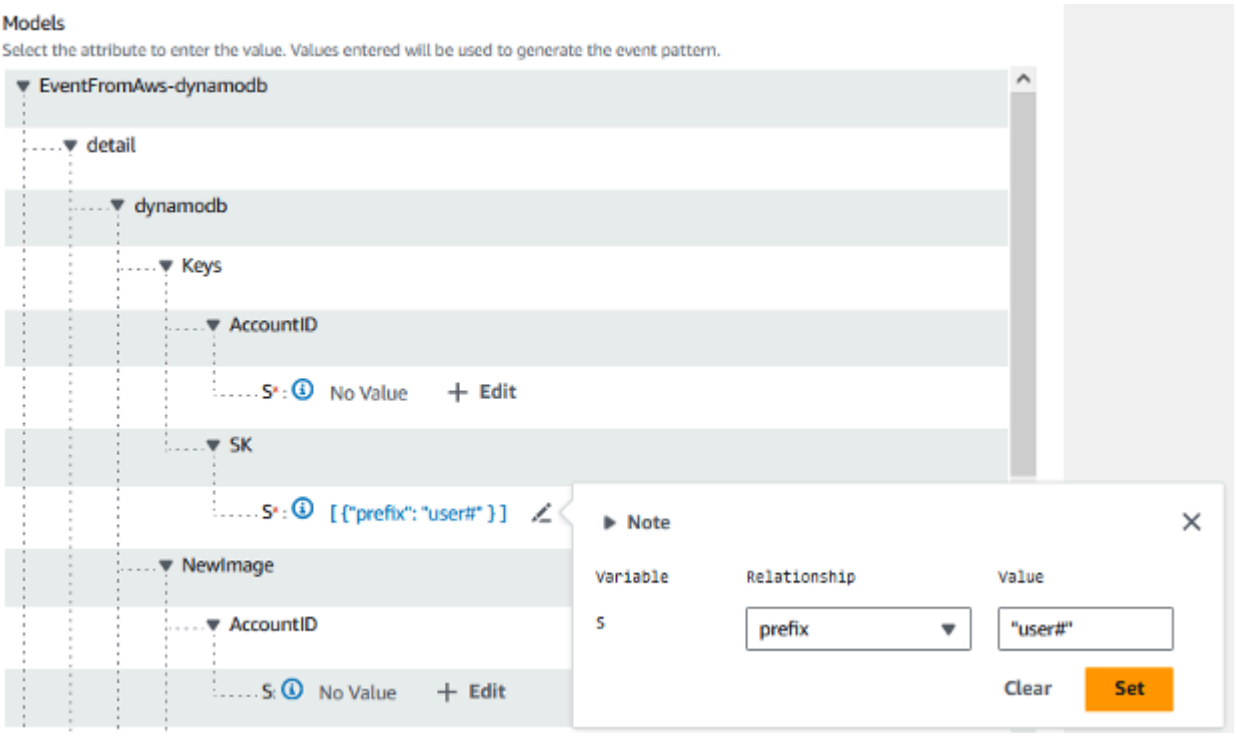
para demostrar la integración. Cuando se invoque, imprimirá la `NewImage` y la `OldImage` que se reciben con el evento y que se pueden ver en Registros de Amazon CloudWatch.

```
import json

def lambda_handler(event, context):
    dynamodb = event.get('detail', {}).get('dynamodb', {})
    new_image = dynamodb.get('NewImage')
    old_image = dynamodb.get('OldImage')

    if new_image:
        print("NewImage:", json.dumps(new_image, indent=2))
    if old_image:
        print("OldImage:", json.dumps(old_image, indent=2))

    return {'statusCode': 200, 'body': json.dumps(event)}
```

6. Cree una regla de EventBridge que enrute los eventos a su nueva función de Lambda. Para ello, siga los pasos de la sección Create a rule that reacts to events de la Guía del usuario de EventBridge.

   a. Cuando defina los detalles de la regla, seleccione el nombre del bus de eventos que creó en el paso 2 como bus de eventos.

   b. Para crear el patrón de eventos, siga la guía del esquema existente. Aquí puede seleccionar el registro de esquemas detectados y el esquema detectado para su evento. Esto le permite configurar un patrón de eventos específico para su caso de uso que solo enrute los mensajes que coincidan con atributos concretos. Por ejemplo, si quisiera que se aplique solo a los elementos de DynamoDB en los que la clave de clasificación (SK) comienza por "`user#`", utilizaría una configuración como esta.



c. Haga clic en Generar patrón de eventos en JSON cuando haya terminado de diseñar un patrón en su esquema. Si, por el contrario, quiere que se aplique a todos los eventos que aparecen en DynamoDB Streams, utilice el siguiente JSON para el patrón de eventos.

```
{
  "source": ["aws.dynamodb"]
}
```

d. Al seleccionar destinos, siga la guía del servicio de AWS. En el campo Seleccionar un destino, elija “Función de Lambda”. En el campo Función, seleccione la función de Lambda que creó en el paso 5.

7. Ahora, para detener la detección de esquemas en su bus de eventos, siga los pasos que se indican la sección Starting or stopping schema discovery on event buses de la Guía del usuario de EventBridge.
8. Escriba un segundo elemento de ejemplo en la tabla de DynamoDB de origen para activar un evento. Valide que el evento se haya procesado correctamente en cada paso.

   a. Consulte la métrica de CloudWatch PutEventsApproximateSuccessCount para su bus de eventos de acuerdo con la sección Monitoring Amazon EventBridge de la Guía del usuario de EventBridge.

   b. Consulte los registros de funciones de su función de Lambda de acuerdo con la sección Supervisión y solución de problemas de funciones de Lambda de la Guía para desarrolladores

de Lambda. Si la función de Lambda usa el código de ejemplo proporcionado, debería ver la `NewImage` y la `OldImage` de DynamoDB Streams impresas en el grupo de registro de Registros de Amazon CloudWatch.

c. Consulte la métrica Recuento de errores e índice de éxito (%) de su función de Lambda en la sección Supervisión y solución de problemas de funciones de Lambda de la Guía para desarrolladores de Lambda.

Siguientes pasos

En este ejemplo se proporciona una integración básica con una sola función de Lambda como destino. Para comprender mejor las configuraciones más complejas, como la creación de varias reglas, la creación de varios destinos, la integración con otros servicios y el enriquecimiento de eventos, consulte la guía completa del usuario de EventBridge: Getting started with EventBridge.

> **Note**
>
> Tenga en cuenta las cuotas de EventBridge que puedan ser relevantes para su aplicación. Aunque la capacidad de DynamoDB Streams se escala con la tabla, las cuotas de EventBridge son independientes. Las cuotas más comunes a tener en cuenta en una aplicación de gran tamaño serían el límite de limitaciones de invocaciones en las transacciones por segundo y el límite de limitaciones de PutEvents en las transacciones por segundo. Estas cuotas especifican el número de invocaciones que se pueden enviar a los destinos y el número de eventos que se pueden incluir en el bus por segundo.

Integración de DynamoDB con Amazon Managed Streaming para Apache Kafka

Amazon Managed Streaming para Apache Kafka (Amazon MSK) facilita la ingesta y el procesamiento de datos de streaming en tiempo real con un servicio de Apache Kafka completamente administrado y de alta disponibilidad.

Apache Kafka es un almacén de datos distribuido optimizado para ingerir y procesar datos de streaming en tiempo real. Kafka puede procesar flujos de registros, almacenar de forma efectiva flujos de registros en el orden en que se generaron y publicar y suscribirse a flujos de registros.

Gracias a estas características, Apache Kafka se suele utilizar para crear canalizaciones de datos de streaming en tiempo real. Una canalización de datos procesa y traslada los datos de un sistema a otro de forma fiable y puede ser una parte importante de la adopción de una estrategia de base de datos personalizada, ya que facilita el uso de varias bases de datos que admiten diferentes casos de uso.

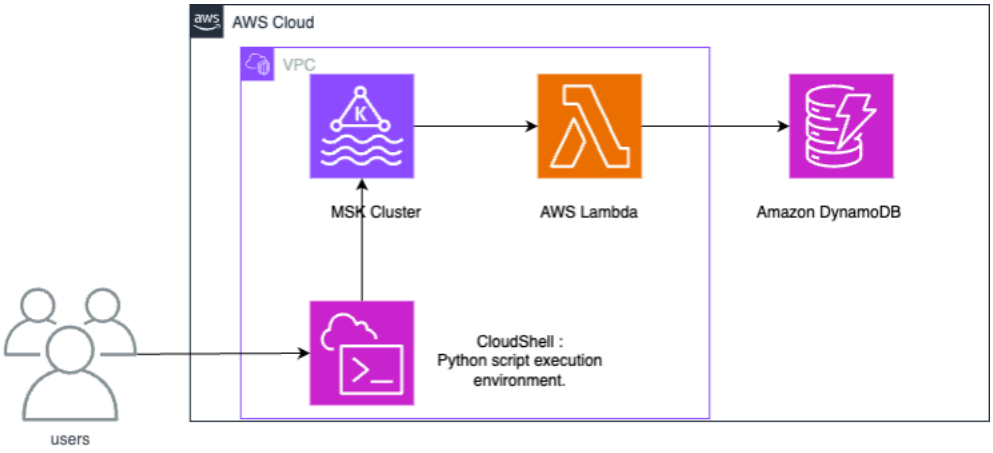
Amazon DynamoDB es un destino común en estas canalizaciones de datos para respaldar aplicaciones que utilizan modelos de datos de clave-valor o de documentos y desean una escalabilidad ilimitada con un rendimiento constante de milisegundos de un solo dígito.

Temas

- Funcionamiento
- Configuración de una integración entre Amazon MSK y DynamoDB
- Pasos a seguir a continuación

Funcionamiento

Una integración entre Amazon MSK y DynamoDB utiliza una función de Lambda para consumir registros de Amazon MSK y escribirlos en DynamoDB.



Lambda sondea internamente nuevos mensajes de Amazon MSK y, a continuación, invoca de forma sincrónica la función de Lambda de destino. La carga útil de eventos de la función de Lambda contiene lotes de mensajes de Amazon MSK. Para la integración entre Amazon MSK y DynamoDB, la función de Lambda escribe estos mensajes en DynamoDB.

Configuración de una integración entre Amazon MSK y DynamoDB

> **Note**
>
> Puede descargar los recursos utilizados en este ejemplo en el siguiente repositorio de GitHub.

En los pasos siguientes se muestra cómo establecer una integración de muestra entre Amazon MSK y Amazon DynamoDB. En el ejemplo se representan datos generados por dispositivos del Internet de las cosas (IoT) e ingeridos en Amazon MSK. A medida que se realiza la ingesta de datos en Amazon MSK, estos pueden integrarse con servicios de análisis o herramientas de terceros compatibles con Apache Kafka, lo que habilita diversos casos de uso de análisis. La integración de DynamoDB también proporciona la búsqueda de valores clave de los registros de dispositivos individuales.

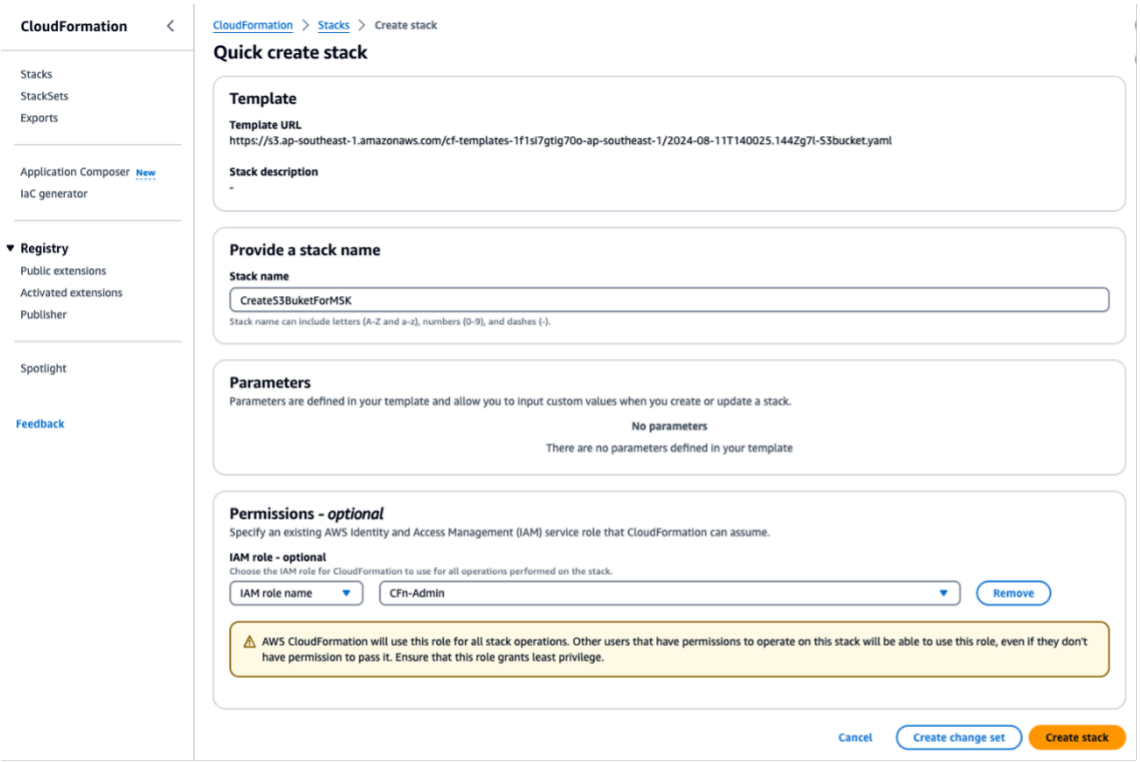
En este ejemplo se demostrará cómo un script de Python escribe datos de sensores IoT en Amazon MSK. A continuación, una función de Lambda escribe los elementos con la clave de partición "`deviceid`" en DynamoDB.

La plantilla de CloudFormation proporcionada creará los siguientes recursos: un bucket de Amazon S3, una Amazon VPC, un clúster de Amazon MSK y un AWS CloudShell para probar las operaciones de datos.

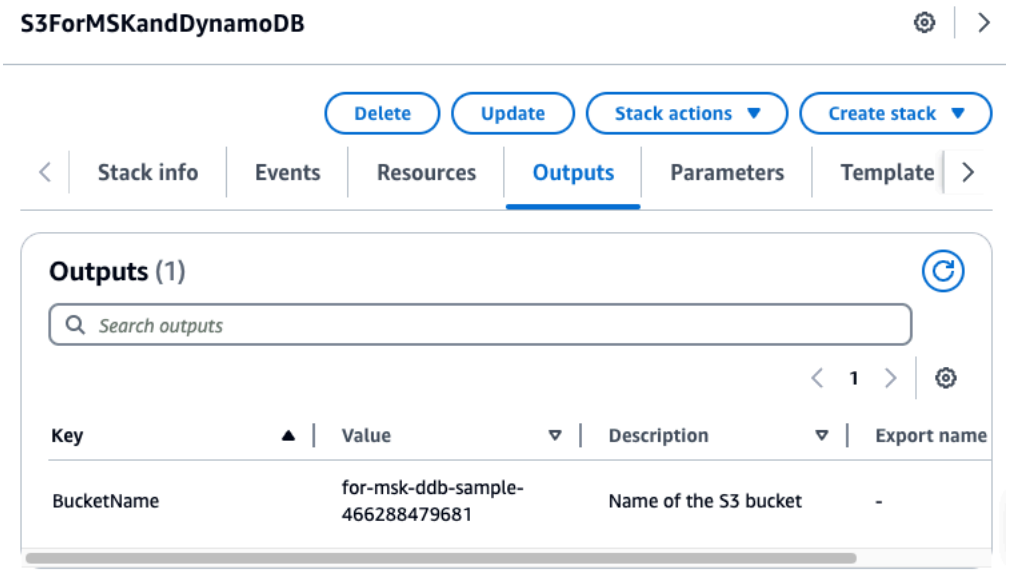
Para generar los datos de prueba, cree un tema de Amazon MSK y, a continuación, cree una tabla de DynamoDB. Puede utilizar Session Manager desde la consola de administración para iniciar sesión en el sistema operativo de CloudShell y ejecutar scripts de Python.

Después de ejecutar la plantilla de CloudFormation, puede terminar de crear esta arquitectura mediante las siguientes operaciones.

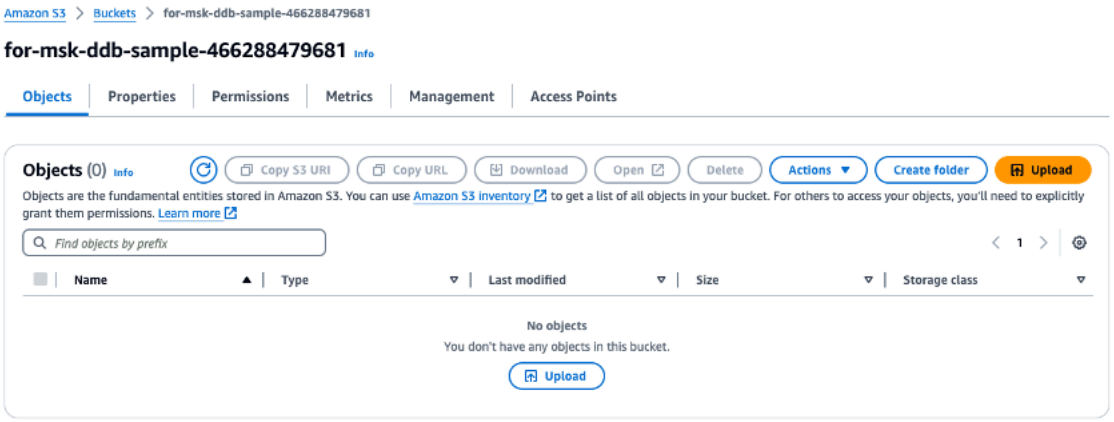
1. Ejecute la plantilla de CloudFormation `S3bucket.yaml` para crear un bucket de S3. Para cualquier script u operación posterior, ejecútelos en la misma región. Introduzca `ForMSKTestS3` como el nombre de la pila de CloudFormation.



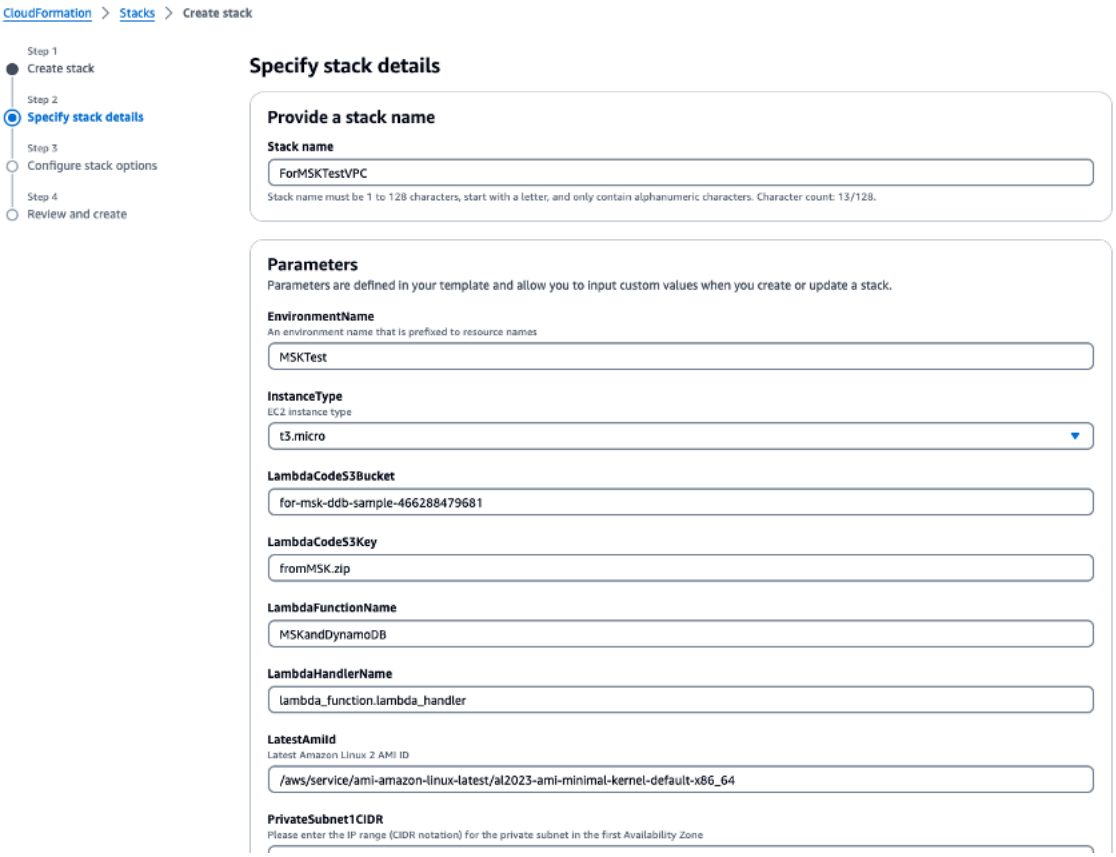
Una vez hecho esto, anote el nombre de la salida del bucket de S3 en Salidas. Necesitará el nombre en el paso 3.



2. Cargue el archivo ZIP descargado `fromMSK.zip` en el bucket de S3 que acaba de crear.



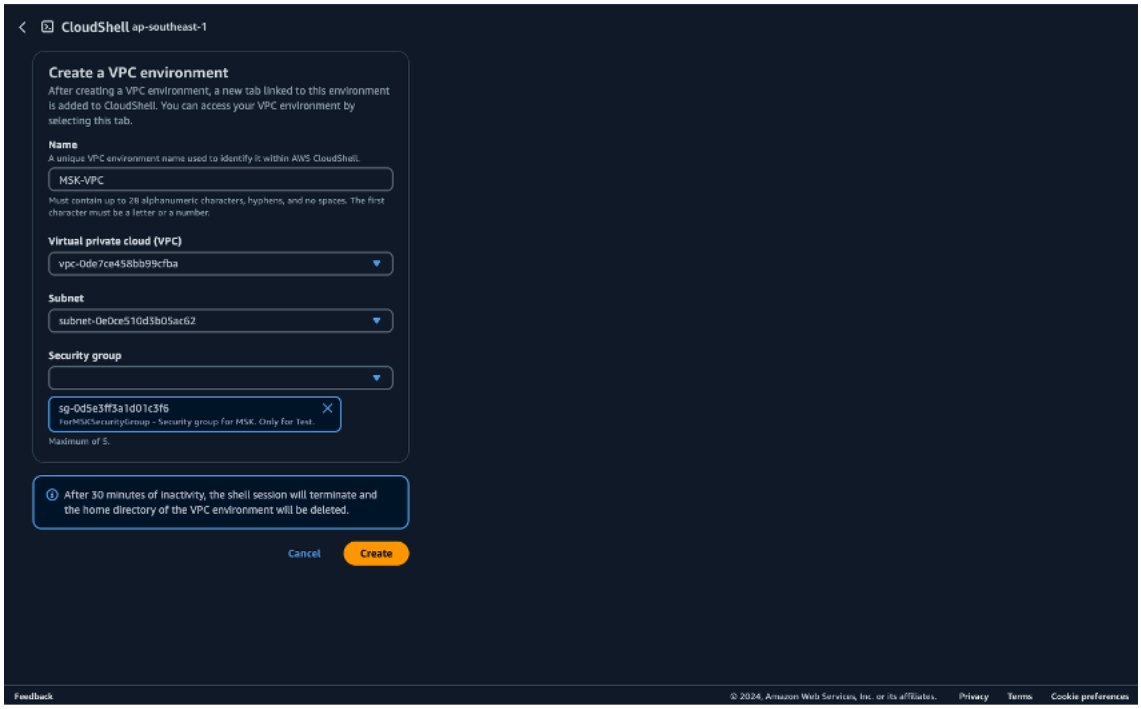
3. Ejecute la plantilla de CloudFormation `VPC.yaml` para crear una VPC, un clúster de Amazon MSK y una función de Lambda. En la pantalla de introducción de parámetros, introduzca el nombre del bucket de S3 que ha creado en el paso 1, donde se solicita el bucket de S3. Establezca el de la pila de CloudFormation a `ForMSKTestVPC`.



4. Prepare el entorno para ejecutar scripts de Python en CloudShell. Puede usar CloudShell en la Consola de administración de AWS. Para obtener más información sobre el uso de CloudShell, consulte Introducción a AWS CloudShell. Después de iniciar CloudShell, cree un CloudShell que

pertenezca a la VPC que acaba de crear para conectarse al clúster de Amazon MSK. Cree el CloudShell en una subred privada. Rellene los siguientes campos:

1. Nombre: se puede establecer a cualquier nombre. Un ejemplo es MSK-VPC
2. VPC: seleccione MSKTest
3. Subred: seleccione Subred privada MSKTest (AZ1)
4. SecurityGroup: seleccione ForMSKSecurityGroup



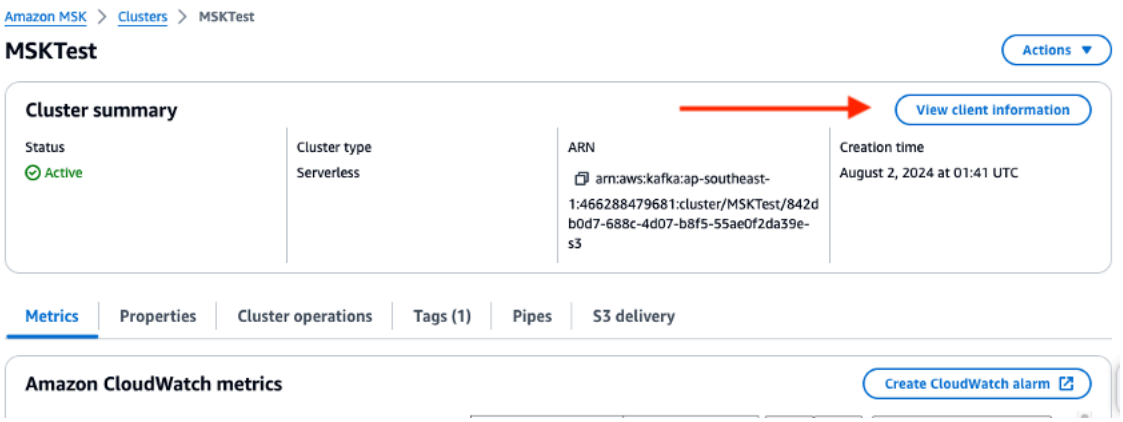
Una vez iniciado el CloudShell perteneciente a la subred privada, ejecute el siguiente comando:

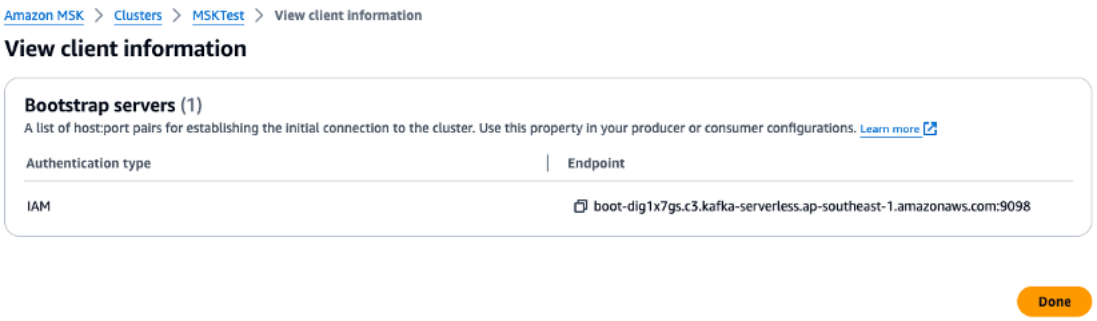
```
pip install boto3 kafka-python aws-msk-iam-sasl-signer-python
```

5. Descargue los scripts de Python del bucket de S3.

```
aws s3 cp s3://[YOUR-BUCKET-NAME]/pythonScripts.zip ./
unzip pythonScripts.zip
```

6. Consulte la consola de administración y establezca las variables de entorno para la URL de agente y el valor de región en los scripts de Python. Consulte el punto de conexión de agente de clúster de Amazon MSK en la consola de administración.





7. Establezca las variables de entorno en el CloudShell. Si utiliza la región Oeste de EE. UU. (Oregón):

```
export AWS_REGION="us-west-2"
export MSK_BROKER="boot-YOURMSKCLUSTER.c3.kafka-serverless.ap-
southeast-1.amazonaws.com:9098"
```

8. Ejecute los siguientes scripts de Python.

   Crear un tema de Amazon MSK:

```
python ./createTopic.py
```

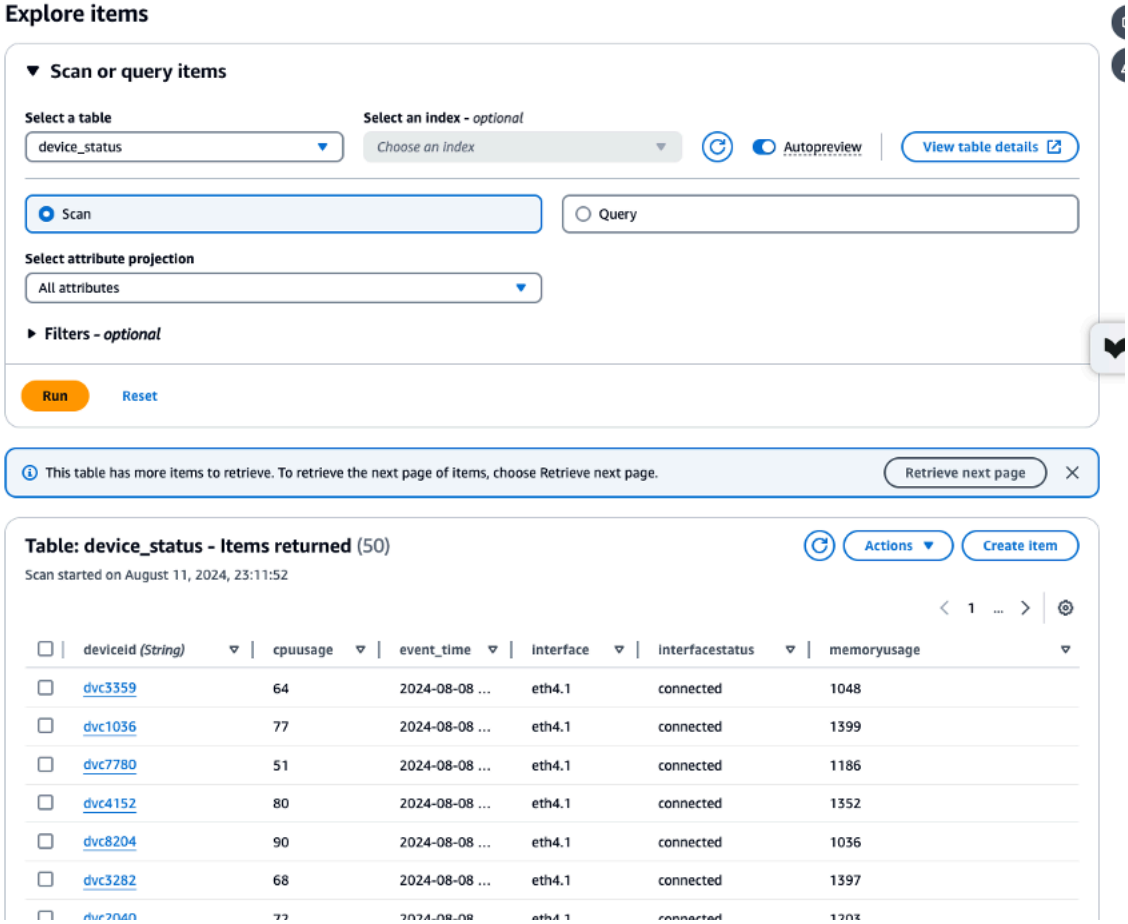
   Crear una tabla de DynamoDB:

```
python ./createTable.py
```

   Escribir los datos de prueba en el tema de Amazon MSK:

```
python ./kafkaDataGen.py
```

9. Consulte las métricas de CloudWatch correspondientes a los recursos creados de Amazon MSK, Lambda y DynamoDB, y verifique los datos almacenados en la tabla `device_status` mediante el explorador de datos de DynamoDB para asegurarse de que todos los procesos se han ejecutado correctamente. Si cada proceso se ejecuta sin errores, puede comprobar si los datos de prueba escritos desde CloudShell a Amazon MSK también se escriben en DynamoDB.



10. Cuando haya terminado con este ejemplo, elimine los recursos creados en este tutorial. Elimine las dos pilas de CloudFormation: `ForMSKTestS3` y `ForMSKTestVPC`. Si la eliminación de la pila se completa de forma correcta, se eliminarán todos los recursos.

Pasos a seguir a continuación

> **Note**
>
> Si ha creado recursos mientras seguía este ejemplo, recuerde eliminarlos para evitar cargos inesperados.

La integración ha identificado una arquitectura que vincula Amazon MSK y DynamoDB para habilitar los datos de flujo a fin de respaldar las cargas de trabajo OLTP. A partir de aquí, se pueden realizar búsquedas más complejas si se vincula DynamoDB con OpenSearch Service. Considere la posibilidad de realizar una integración con EventBridge para necesidades más complejas basadas en

eventos y extensiones como Amazon Managed Service para Apache Flink para requisitos de mayor rendimiento y menor latencia.

Prácticas recomendadas para la integración con DynamoDB

Al integrar DynamoDB con otros servicios, debe seguir siempre las prácticas recomendadas sobre el uso de cada servicio. Además, debe tener en cuenta algunas prácticas recomendadas específicas de la integración.

Temas

- Creación de una instantánea en DynamoDB
- Captura de datos de cambios en DynamoDB

Creación de una instantánea en DynamoDB

- En general, recomendamos utilizar la exportación a Amazon S3 para crear instantáneas para la replicación inicial. Esta opción es rentable y no competirá por el rendimiento con el tráfico de su aplicación. Otra opción es realizar una copia de seguridad, restaurarla en una nueva tabla y finalizar con una operación de análisis. Esto evitará que se compita por el rendimiento con la aplicación, aunque, por lo general, será considerablemente menos rentable que una exportación.
- Defina siempre una `StartTime` al exportar. De este modo, será más fácil determinar dónde debe empezar la captura de datos de cambios (CDC).
- Si utiliza la exportación a S3, establezca una acción de ciclo de vida en el bucket de S3. Por lo general, establecer una acción con una caducidad de siete días es seguro, pero debe seguir las pautas definidas por su empresa. Incluso si elimina los elementos de forma explícita después de la ingesta, esta acción puede ayudar a detectar problemas. De este modo, se reduce cualquier costo innecesario y se evita cualquier infracción de las políticas.

Captura de datos de cambios en DynamoDB

- Si precisa una captura de datos de cambios (CDC) casi en tiempo real, utilice DynamoDB Streams o Amazon Kinesis Data Streams (KDS). A la hora de decidirse por una de las dos opciones, tenga en cuenta cuál es más fácil de usar con el servicio posterior. Si tiene que proporcionar un procesamiento de eventos ordenado en el nivel de la clave de partición, o si tiene unos elementos excepcionalmente grandes, utilice DynamoDB Streams.

- Si no necesita una CDC prácticamente en tiempo real, puede utilizar la exportación a Amazon S3 con exportaciones incrementales para exportar solo los cambios que se hayan producido entre dos momentos concretos.

  Si ha utilizado la exportación a S3 para generar una instantánea, puede resultar especialmente útil, ya que puede utilizar un código similar para procesar las exportaciones incrementales. Por lo general, la exportación a S3 es un poco más económica que las opciones de secuencia anteriores, pero a la hora de elegir una opción, el costo no suele ser el factor primordial.
- En general, solo puede haber dos consumidores simultáneos de una secuencia de DynamoDB. Tenga esto en cuenta a la hora de planificar su estrategia de integración.
- No utilice análisis para detectar cambios. Esto puede funcionar a pequeña escala, pero resulta ser poco práctico de forma bastante rápida.

Uso de la IA generativa con DynamoDB

Amazon DynamoDB es una base de datos NoSQL distribuida sin servidor completamente administrada con un rendimiento de milisegundos de un solo dígito a cualquier escala. DynamoDB se ha optimizado para cargas de trabajo de alto rendimiento y puede ampliar las capacidades mediante la integración con modelos de IA generativa. Al utilizar modelos de IA generativa, puede trabajar con datos almacenados en tablas de DynamoDB en tiempo real y crear aplicaciones que tengan en cuenta el contexto y estén muy personalizadas. También puede mejorar la experiencia del usuario final si aprovecha al máximo los datos empresariales, así como de usuario y de aplicación para personalizar las soluciones de IA generativa.

Para obtener más información sobre la IA de generación y las soluciones que AWS ofrece para crear aplicaciones de IA de generación, consulte Transforme su negocio gracias la IA generativa.

Temas

- Casos de uso de IA generativa para DynamoDB
- Blogs de IA generativa para DynamoDB
- Uso de la integración sin ETL de DynamoDB con el servicio OpenSearch

Casos de uso de IA generativa para DynamoDB

DynamoDB tiene un uso muy extendido en aplicaciones conversacionales basadas en IA, como chatbots y centros de llamadas creados con un modelo fundacional (FM). Puede acceder a los FM a través de Amazon Bedrock, Amazon SageMaker AI u otros proveedores de modelos. Estas aplicaciones suelen utilizar DynamoDB para mejorar la personalización y la experiencia del usuario en tres patrones de datos: aplicación, empresa y usuario. A continuación, se ofrecen algunos ejemplos de estos patrones de datos:

- Almacenamiento de datos de aplicación, como el historial de mensajes de chat, mediante integraciones con LangChain, LlamaIndex o un código personalizado. Este contexto mejora la experiencia del usuario al permitir que el modelo converse con él.
- Creación de una experiencia de usuario personalizada mediante el uso de datos empresariales, como el inventario, los precios y la documentación.
- Aplicación de datos de usuario, como el historial web, los pedidos anteriores y las preferencias del usuario, para ofrecer respuestas personalizadas.

Por ejemplo, una compañía de seguros puede crear un chatbot que utilice DynamoDB para proporcionar acceso a datos casi en tiempo real al modelo de IA generativa basado en la generación aumentada por recuperación (RAG). Ejemplos de estos datos son los tipos de interés hipotecarios en tiempo real, los precios de productos, la copia de contratos conformes o estándar, el historial web del usuario y las preferencias de usuario. La combinación de DynamoDB con RAG agrega información detallada y actualizada sobre los productos de seguros y los datos de usuario. Esto enriquece las peticiones y las respuestas para ofrecer a los usuarios finales una experiencia precisa, personalizada y casi en tiempo real.

Del mismo modo, los clientes del sector de servicios financieros utilizan DynamoDB, las bases de conocimientos de Amazon Bedrock y los agentes de Amazon Bedrock para crear aplicaciones de IA generativa basadas en RAG. Estas aplicaciones pueden utilizar informes de ingresos de código abierto y transcripciones de llamadas. También pueden utilizar la cartera específica del usuario y el historial de transacciones para generar un resumen bajo demanda de la cartera que incluya una previsión de futuro.

Blogs de IA generativa para DynamoDB

En los siguientes artículos se ofrecen casos de uso detallados, prácticas recomendadas y guías paso a paso que lo ayudarán a aprovechar las capacidades de DynamoDB para crear aplicaciones avanzadas basadas en IA.

- Amazon DynamoDB data models for generative AI chatbots
- Build a scalable, context-aware chatbot with Amazon DynamoDB, Amazon Bedrock, and LangChain

Uso de la integración sin ETL de DynamoDB con el servicio OpenSearch

Puede utilizar Amazon Bedrock con DynamoDB para proporcionar acceso sin servidor a modelos fundacionales (FM), como Amazon Titan y otros modelos de terceros. Puede aprovechar la integración sin ETL con Amazon OpenSearch Service para habilitar las capacidades de búsqueda vectorial al crear aplicaciones de IA generativa. El taller Generative AI with DynamoDB zero-ETL to OpenSearch integration and Amazon Bedrock le proporciona experiencia práctica en la configuración de la integración sin ETL de DynamoDB con OpenSearch. Este taller realiza las siguientes tareas:

- Crea una canalización desde la tabla de DynamoDB a OpenSearch.

- Crea un conector de Amazon Bedrock en OpenSearch.
- Consulta Amazon Bedrock mediante el uso de OpenSearch como almacén vectorial.
- Utiliza el FM Claude en Amazon Bedrock para crear una respuesta redactada en lenguaje sencillo que explique los resultados de búsqueda que devuelve OpenSearch.

Este taller le permite integrar DynamoDB con OpenSearch para crear aplicaciones de IA generativa. También demuestra la capacidad de consulta flexible entre motores de bases de datos para ayudarlo a integrar DynamoDB y OpenSearch en casos de uso tradicionales. Este taller es uno de los siete módulos de Amazon DynamoDB Immersion Day. Puede realizar este taller en cualquier Cuenta de AWS.

También puede consultar la siguiente entrada de blog sobre cómo establecer una integración sin ETL entre DynamoDB y OpenSearch Service. En esta entrada de blog también se describe cómo establecer conectores de modelos en OpenSearch Service para generar de forma automática incrustaciones con Amazon Bedrock para los datos entrantes. Vector search for Amazon DynamoDB with zero ETL for Amazon OpenSearch Service.

Cuotas y restricciones para Amazon DynamoDB

En este tema se describen las cuotas actuales, anteriormente se denominaban límites, de Amazon DynamoDB. En este tema también se describe cómo puede realizar las tareas de administración de cuotas, por ejemplo, ver las cuotas actuales y solicitar un aumento de las cuotas.

Temas

- Realización de tareas de administración de cuotas en DynamoDB
- Solicitud de un aumento de cuota en DynamoDB
- Cuotas en Amazon DynamoDB
- Restricciones en Amazon DynamoDB

Realización de tareas de administración de cuotas en DynamoDB

Amazon DynamoDB tiene varios componentes de servicio, como tablas, transmisiones, índices, etc. Al crear la Cuenta de AWS, se establecen cuotas predeterminadas (anteriormente denominadas límites) para estos componentes. A menos que se indique lo contrario, cada cuota es específica de la región. Puede solicitar aumentos de algunas de las cuotas. Una vez que se alcance la cuota de un recurso, las solicitudes adicionales para crear ese recurso dejan de funcionar con una excepción.

Acceso a las cuotas de DynamoDB

Puede trabajar con Service Quotas de DynamoDB de las siguientes maneras:

- Consola de administración de AWS

  La consola de Service Quotas es una interfaz basada en navegador que puede utilizar para ver y administrar las Service Quotas. Puede acceder a Service Quotas desde cualquier página de Consola de administración de AWS seleccionándola en la barra de navegación superior o buscando Service Quotas en Consola de administración de AWS.
- Herramientas de AWS Command Line Interface

  Con las herramientas de AWS Command Line Interface, puede emitir comandos en la línea de comandos del sistema para llevar a cabo tareas de Service Quotas. Las herramientas de la línea de comandos son útiles si desea crear scripts que lleven a cabo tareas de AWS.
- AWS SDK

Puede utilizar los SDK de AWS para diversos lenguajes de programación y plataformas (por ejemplo, Java, Python, Ruby, .NET, iOS y Android, entre otros) para llevar a cabo tareas de Service Quotas.

Si una cuota ajustable no está disponible en la consola de Service Quotas, utilice la AWS Support Center Console para crear un caso de incremento de Service Quotas.

Visualización de las cuotas actuales en la consola

Visualización de las cuotas actuales de DynamoDB con la consola de Service Quotas

1. Apertura de la consola de Service Quotas en https://console.aws.amazon.com/servicequotas/home/services/dynamodb/quotas/
2. Desde la barra de navegación, parte superior de la pantalla, seleccione una región.
3. La consola muestra detalles sobre el nombre de la cuota de DynamoDB, el valor de cuota en el nivel de cuenta aplicado, el valor de cuota predeterminado de AWS, la utilización y la capacidad de ajuste de la cuota en el nivel de cuenta o de recursos.

   Si el valor de la cuota o la utilización aplicados no están disponibles, la consola muestra No disponible. Puede solicitar el valor de la cuota aplicada a través de la consola del Centro de soporte.
4. Elija un nombre de cuota específico para ver la página de detalles, que muestra la descripción de la cuota, el código de cuota, el ARN de la cuota, la utilización, el valor de cuota aplicado en el nivel de cuenta, la capacidad de ajuste y el valor de cuota predeterminado de AWS.

   Si corresponde, la página de detalles también muestra las opciones de supervisión, las alarmas, el historial de solicitudes y cualquiera de las etiquetas de la cuota.

Visualización de las cuotas actuales mediante AWS CLI

Para ver los valores predeterminados de las cuotas de DynamoDB:

- Llame a la operación `ListDefaultServiceQuotas` con el código de servicio de DynamoDB (dynamodb) para recuperar los valores predeterminados de las Service Quotas de Amazon DynamoDB.

```
$ aws service-quotas list-aws-default-service-quotas \
        --service-code dynamodb

 {
    "Quotas": [
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1::dynamodb/L-F7858A77",
            "QuotaCode": "L-F7858A77",
            "QuotaName": "Global Secondary Indexes per table",
            "Value": 20.0,
            "Unit": "None",
            "Adjustable": true,
            "GlobalQuota": false
        },
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1::dynamodb/L-AB614373",
            "QuotaCode": "L-AB614373",
            "QuotaName": "Table-level write throughput limit",
            "Value": 40000.0,
            "Unit": "None",
            "Adjustable": true,
            "GlobalQuota": false
        }......
    ]
}
```

Para ver los valores de cuota aplicados:

- Llame a la operación `ListServiceQuotas` con el código de servicio de DynamoDB (dynamodb) para recuperar todos los valores de cuota aplicados en el nivel de cuenta, nivel de recursos o todos los niveles al pasar `ACCOUNT`, `RESOURCE` o `ALL`, respectivamente, como valor del parámetro `QuotaAppliedAtLevel`. En el siguiente ejemplo de la CLI se recuperan los valores de cuota aplicados en el nivel de cuenta.

```
$ aws service-quotas list-service-quotas \
        --service-code dynamodb \
```

```
        --quota-applied-at-level ACCOUNT


{
    "Quotas": [
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1:303935678045:dynamodb/L-
F7858A77",
            "QuotaCode": "L-F7858A77",
            "QuotaName": "Global Secondary Indexes per table",
            "Value": 20.0,
{
    "Quotas": [
        {
            "ServiceCode": "dynamodb",
            "ServiceName": "Amazon DynamoDB",
            "QuotaArn": "arn:aws:servicequotas:us-east-1:303935678045:dynamodb/L
-F7858A77",
            "QuotaCode": "L-F7858A77",
            "QuotaName": "Global Secondary Indexes per table",
            "Value": 20.0,
            "Unit": "None",
            "Adjustable": true,
            "GlobalQuota": false,
            "QuotaAppliedAtLevel": "ACCOUNT"
        }.....

        }
    ]
}
```

Solicitud de un aumento de cuota en DynamoDB

Puede solicitar un aumento de cuota para cada región mediante la consola de Service Quotas, AWS CLI o un caso de soporte. Si una cuota ajustable no está disponible en la consola de Service Quotas, utilice la AWS Support Center Console para crear un caso de incremento de Service Quotas.

Soporte podría aprobar, denegar o aprobar parcialmente las solicitudes de aumento de cuota. Los aumentos no se conceden de inmediato y pueden tardar varios días en surtir efecto.

Para solicitar un aumento, visite la Consola de Service Quotas.

1. Abra la consola de Service Quotas en https://console.aws.amazon.com/servicequotas/home/services/dynamodb/quotas/
2. Desde la barra de navegación, parte superior de la pantalla, seleccione una región.
3. Filtre la lista por nombre de recurso. Por ejemplo, ingrese Bajo demanda para localizar las cuotas de las instancias bajo demanda.
4. Si la cuota es ajustable, selecciónela y elija Solicitar aumento de cuota.
5. En Cambiar valor de cuota, ingrese el nuevo valor.
6. Seleccione Request (Solicitar).
7. Para ver las solicitudes pendientes o resueltas recientemente, elija Panel en el panel de navegación. Para las solicitudes pendientes, seleccione el estado de la solicitud para abrir la recepción de solicitud. El estado inicial de una solicitud es Pendiente. Después de que el estado cambie a Cuota solicitada, verá el número de caso con Soporte. Elija el número de caso para abrir el ticket para su solicitud.

Para obtener más información, incluida la forma de utilizar la AWS CLI o los SDK o para solicitar un aumento de cuota, consulte Solicitud de un aumento de cuota en la Guía del usuario de Service Quotas.

Cuotas en Amazon DynamoDB

En esta sección se describen las cuotas actuales (anteriormente se denominaban límites) de Amazon DynamoDB. Cada una de las cuotas se aplica a una sola región, a no ser que se especifique otra cosa.

Temas

- Rendimiento de lectura o escritura
- Capacidad reservada
- Tablas
- Tablas globales
- Índices secundarios
- Atributos de índice secundario proyectados

- DynamoDB Streams
- Importación desde Amazon S3
- Exportar tablas a Amazon S3
- Copia de seguridad y restauración
- Contributor Insights

Rendimiento de lectura o escritura

Cuotas de rendimiento predeterminadas

AWS aplica unas cuotas predeterminadas al rendimiento que su cuenta puede aprovisionar y consumir en una región.

El rendimiento de lectura en el nivel de cuenta y las cuotas de rendimiento de escritura en el nivel de cuenta se aplican en el nivel de cuenta. Estas cuotas en el nivel de cuenta se aplican a la suma de la capacidad de rendimiento aprovisionada para todas las tablas de cuentas e índices secundarios globales de una región determinada. Todo el rendimiento disponible para la cuenta se puede aprovisionar para una sola tabla o para varias tablas. Estas cuotas solo se aplican a las tablas que utilizan el modo de capacidad aprovisionada.

Las cuotas de rendimiento de lectura en el nivel de tabla y de rendimiento de escritura en el nivel de tabla se aplican de forma diferente a las tablas que utilizan el modo de capacidad aprovisionada y a las tablas que utilizan el modo de capacidad bajo demanda.

Para las tablas y los GSI en modo de capacidad aprovisionada, la cuota es la cantidad máxima de unidades de capacidad de lectura y escritura que se pueden aprovisionar para cualquier tabla o cualquiera de sus GSI en la región. El total de cualquier tabla individual y de todos sus GSI también debe permanecer por debajo de la cuota de rendimiento de lectura y escritura en el nivel de cuenta. Esto se agrega al requisito de que el total de todas las tablas aprovisionadas y sus GSI deben permanecer por debajo de la cuota de rendimiento de lectura y escritura a nivel de cuenta.

Para las tablas y los GSI en modo de capacidad bajo demanda, la cuota en el nivel de tabla es el máximo de unidades de capacidad de lectura y escritura que están disponibles para cualquier tabla, o cualquier GSI individual en dicha tabla. No se aplican cuotas de rendimiento de lectura y escritura en el nivel de cuenta a las tablas en modo bajo demanda.

Estas son las cuotas de rendimiento que se aplican a su cuenta de forma predeterminada.

> **Note**
>
> Puede solicitar cualquier número de unidades de capacidad de lectura (RCU) o unidades de capacidad de escritura (WCU) para las tablas de DynamoDB mediante un aumento de la cuota de servicio. Los valores enumerados en la siguiente tabla representan las cuotas iniciales predeterminadas. No se trata de límites máximos para las tablas.

Nombre de la cuota de rendimiento	Bajo demanda	Aprovisionado	Ajustable
Por tabla	40 000 unidades de solicitud de lectura y 40 000 unidades de solicitud de escritura	40 000 unidades de capacidad de lectura y 40 000 unidades de capacidad de escritura	Sí
Por cuenta de	No aplicable	80 000 unidades de capacidad de lectura y 80 000 unidades de capacidad de escritura	Sí
Rendimiento mínimo para cualquier tabla o índice secundario global	No aplicable	1 unidad de capacidad de lectura y 1 unidad de capacidad de escritura	Sí

Aumento o reducción del rendimiento (para las tablas aprovisionadas)

Aumento del rendimiento aprovisionado

Puede aumentar el valor de `ReadCapacityUnits` o `WriteCapacityUnits` con tanta frecuencia como sea preciso; para ello, puede usar la Consola de administración de AWS o la operación `UpdateTable`. En una sola llamada, puede aumentar el rendimiento aprovisionado de una tabla,

de cualquier índice secundario global de esa tabla o de cualquier combinación de ellos. La nueva configuración no surtirá efecto hasta que se haya completado la operación `UpdateTable`.

No puede superar las cuotas por cuenta al agregar capacidad aprovisionada. DynamoDB no permite aumentar la capacidad aprovisionada con gran rapidez. Aparte de estas restricciones, puede aumentar la capacidad aprovisionada de las tablas tanto como lo necesite. Para obtener más información sobre las cuotas por cuenta, consulte la sección anterior, Cuotas de rendimiento predeterminadas.

Reducción de rendimiento aprovisionado

En cada tabla e índice secundario global de una operación `UpdateTable`, puede reducir el valor de `ReadCapacityUnits`, de `WriteCapacityUnits` o de ambas opciones. La nueva configuración no surtirá efecto hasta que se haya completado la operación `UpdateTable`.

Existe una cuota predeterminada del número de reducciones de capacidad aprovisionada que puede realizar en su tabla de DynamoDB por día. Un día se define según la hora universal coordinada (UTC). En un día determinado, puede empezar realizando hasta cuatro disminuciones en una hora, siempre que no haya realizado todavía ninguna otra disminución durante ese día. Posteriormente, puede realizar una disminución adicional por hora (una vez cada 60 minutos). De hecho, esto eleva el número máximo de disminuciones en un día a 27 veces.

> **Important**
>
> Los límites de reducción de tablas e índices secundarios globales no están asociados, lo que significa que los índices secundarios globales de una determinada tabla tienen sus propios límites de reducción. Sin embargo, si una solicitud reduce el rendimiento de una tabla y un índice secundario global, se rechazará si se supera alguno de los límites actuales. Las solicitudes no se procesan parcialmente.

Example

En las primeras 4 horas de un día, una tabla con un índice secundario global puede modificarse de la siguiente manera:

- Reduzca los valores `WriteCapacityUnits` o `ReadCapacityUnits` (o ambos) cuatro veces.
- Reduzca los valores `WriteCapacityUnits` o `ReadCapacityUnits` (o ambos) del índice secundario global cuatro veces.

Al final de ese mismo día, el rendimiento de la tabla y del índice secundario global se podría reducir un total de 27 veces cada uno.

Capacidad reservada

AWS establece una cuota predeterminada en la cantidad de capacidad reservada activa que su cuenta puede comprar. El límite de la cuota es una combinación de la capacidad reservada para las unidades de capacidad de escritura (WCU) y las unidades de capacidad de lectura (RCU).

Cuota de capacidad reservada	Capacidad reservada activa	Ajustable
Por cuenta de	1 000 000 de unidades de capacidad aprovisionada (WCU_ RCU)	Sí

Si intenta comprar más de 1 000 000 de unidades de capacidad aprovisionada en una sola compra, recibirá un error con respecto a este límite de cuota de servicio. Si tiene capacidad reservada activa e intenta comprar capacidad reservada adicional, lo que generaría más de 1 000 000 de unidades de capacidad aprovisionadas activas, recibirá un mensaje de error para este límite de cuota de servicio.

Tablas

Tamaño de las tablas

No existe ningún límite práctico del tamaño de una tabla. Las tablas no presentan restricciones en cuanto al número de elementos o de bytes.

Número máximo de tablas por cuenta y región

Para cualquier cuenta de AWS, existe una cuota inicial de 2500 tablas por región de AWS.

Si necesita más de 2500 tablas para una sola cuenta, póngase en contacto con su equipo de cuentas de AWS para estudiar un aumento hasta un máximo de 10 000 tablas. Para más de 10 000, la práctica recomendada es configurar varias cuentas, cada una de las cuales puede servir hasta 10 000 tablas.

Tablas globales

Cuando se utilizan tablas globales, se aplican las siguientes cuotas predeterminadas.

Cuotas de la tabla global predeterminada	Bajo demanda	Aprovisionado
Número de tablas globales MRSC (consulte the section called “Modos de coherencia”)	400 tablas globales MRSC en total en cualquier modo de capacidad	400 tablas globales MRSC en total en cualquier modo de capacidad
Rendimiento por tabla configurado para coherenci a final de varias regiones (MREC)	40 000 unidades de solicitud de lectura y 40 000 unidades de solicitud de escritura	40 000 unidades de capacidad de lectura y 40 000 unidades de capacidad de escritura
Rendimiento por tabla configurado para consisten cia sólida de varias regiones (MRSC)	40 000 unidades de solicitud de lectura y 40 000 unidades de solicitud de escritura	40 000 unidades de capacidad de lectura y 40 000 unidades de capacidad de escritura
Datos rellenados para nuevas réplicas por cuenta, región y día	10 TB	10 TB

Note

Puede haber casos en los que necesite solicitar un aumento del límite de cuota a través de AWS Support. Si se encuentra en alguna de las siguientes circunstancias, consulte https://aws.amazon.com/support:

- Las cuotas de rendimiento de tablas globales deben ser iguales o superiores a las cuotas de rendimiento por tabla para que la creación de réplicas se realice correctamente. Existen cuotas de rendimiento de tablas globales independientes para las tablas globales MREC y MRSC.
- Si va a agregar una réplica o réplicas a una región de destino en un plazo de 24 horas con un total combinado superior a 10 TB, debe solicitar un aumento de la cuota de servicio para la cuota de reposición de datos de réplica agregada.
- Si obtiene un error similar al siguiente:

- No se puede crear una réplica de la tabla “tabla_de_ejemplo” en la región “región_de_ejemplo_A” porque supera el límite de su cuenta actual en la región “región_de_ejemplo_B”.

Índices secundarios

Puede definir hasta 5 índices secundarios globales por tabla.

Existe una cuota predeterminada de 20 índices secundarios globales por tabla.

Atributos de índice secundario proyectados

Puede proyectar hasta 100 atributos combinados para todos los índices secundarios locales y globales de una tabla. Esta cuota solo se aplica a los atributos proyectados especificados por el usuario.

Para la operación `CreateTable`, si especifica un `ProjectionType` como valor de `INCLUDE`, el recuento total de atributos especificados en `NonKeyAttributes` y sumados para todos los índices secundarios, no deberá superar el valor de 100. La proyección del mismo nombre de atributo en dos índices diferentes cuenta como dos atributos distintos para la cuota.

Esta cuota no se aplica a los índices secundarios con un `ProjectionType` de `KEYS_ONLY` o `ALL`.

DynamoDB Streams

Lectores simultáneos de una partición en DynamoDB Streams

En el caso de las tablas de una sola región que no sean tablas globales, puede diseñar hasta dos procesos simultáneos para leer desde la misma partición de DynamoDB Streams al mismo tiempo. Si excede este límite, puede producirse una limitación controlada de las solicitudes. En el caso de las tablas globales, le recomendamos que limite el número de lectores simultáneos a uno para evitar la limitación de solicitudes.

Capacidad de escritura máxima de una tabla con DynamoDB Streams habilitado

AWS establece algunas cuotas predeterminadas respecto a la capacidad de escritura de las tablas de DynamoDB que tienen habilitados flujos de DynamoDB Streams. Estas cuotas predeterminadas solo se aplican a tablas en el modo de capacidad de lectura o escritura aprovisionada.

- Por tabla: 40 000 unidades de capacidad de escritura

Importación desde Amazon S3

La importación de DynamoDB desde Amazon S3 puede admitir hasta 50 trabajos de importación simultáneos con un tamaño total de objetos de origen de importación de 15 TB a la vez en las regiones us-east-1, us-west-2 y eu-west-1. En todas las demás regiones, se admiten hasta 50 tareas de importación simultáneas con un tamaño total de 1 TB. Cada trabajo de importación admite un máximo de 50 000 objetos de Amazon S3 en todas las regiones. Para obtener más información sobre importación y validación, consulte Cuotas de formato de importación y validación.

Exportar tablas a Amazon S3

Exportación completa: se pueden exportar hasta 300 tareas de exportación simultáneas o un total de 100 TB de todas las exportaciones de tablas en proceso. Ambos límites se comprueban antes de poner en cola una exportación.

Exportación incremental: la exportación incremental de DynamoDB a Amazon S3 puede admitir hasta 300 trabajos de exportación simultáneos o hasta un total de 100 TB desde todas las exportaciones de tablas en tránsito. Los límites del periodo de exportación son de 15 minutos como mínimo y 24 horas como máximo.

Copia de seguridad y restauración

DynamoDB admite hasta 50 restauraciones simultáneas por un total de 50 TB a través de copias de seguridad bajo demanda o continuas de DynamoDB. AWS Backup admite hasta 50 restauraciones simultáneas por un total de 25 TB.

Contributor Insights

Al activar Consumer Insights en la tabla de DynamoDB, se seguirán aplicando los límites de las reglas de Contributor Insights. Para obtener más información, consulte CloudWatch Service Quotas.

Restricciones en Amazon DynamoDB

En esta sección se describen las restricciones actuales, anteriormente se denominaban límites, de Amazon DynamoDB.

Temas

- Modo de capacidad de lectura/escritura
- Índices secundarios
- Claves de partición y claves de clasificación
- Reglas de nomenclatura
- Tipos de datos
- Items
- Atributos
- Parámetros de expresión
- Transacciones de DynamoDB
- DynamoDB Streams
- DynamoDB Accelerator (DAX)
- Restricciones específicas de la API
- Cifrado en reposo en DynamoDB

Modo de capacidad de lectura/escritura

Puede cambiar las tablas del modo de capacidad aprovisionada al modo bajo demanda hasta cuatro veces en un periodo continuo de 24 horas. Las tablas pueden cambiar del modo bajo demanda al modo de capacidad aprovisionada en cualquier momento.

Para obtener más información sobre el cambio entre los modos de capacidad de lectura y escritura, consulte Aspectos a tener en cuenta al cambiar los modos de capacidad en DynamoDB.

Tamaños de las unidades de capacidad (para las tablas aprovisionadas)

Una unidad de capacidad de lectura equivale a una lectura de consistencia alta por segundo, o bien a dos lecturas eventualmente consistentes por segundo, para elementos con un tamaño de hasta 4 KB.

Una unidad de capacidad de escritura equivale a una escritura por segundo para los elementos con un tamaño de hasta 1 KB.

Las solicitudes de lectura transaccionales requieren dos unidades de capacidad de lectura por segundo respecto a los elementos de hasta 4 KB.

Las solicitudes de escritura transaccionales requieren dos unidades de capacidad de escritura para realizar una escritura por segundo respecto a los elementos de hasta 1 KB.

Tamaños de las unidades de solicitud (para las tablas bajo demanda)

Una unidad de solicitud de lectura = una lectura altamente coherente por segundo, o dos lecturas coherentes posteriores por segundo, para elementos de hasta 4 KB de tamaño.

Una unidad de solicitud de escritura = una escritura por segundo, para elementos de hasta 1 KB de tamaño.

Las solicitudes de lectura transaccional requieren dos unidades de solicitud de lectura para realizar una lectura por segundo para elementos de hasta 4 KB.

Las solicitudes de escritura transaccionales requieren dos unidades de solicitud de escritura para realizar una escritura por segundo respecto a los elementos de hasta 1 KB.

Índices secundarios

Atributos de índice secundario proyectados por tabla

Puede proyectar un máximo de 100 atributos en todos los índices secundarios locales y globales de una tabla. Esto solo se aplica a los atributos proyectados especificados por el usuario.

En una operación `CreateTable`, si especifica `ProjectionType` como valor de `INCLUDE`, el recuento total de atributos especificados en `NonKeyAttributes` y sumados para todos los índices secundarios, no deberá superar el valor de 100. Si se proyecta el mismo nombre de atributo en dos índices diferentes, esto cuenta como dos atributos distintos a la hora de determinar la cantidad total.

Este límite no se aplica a los índices secundarios cuyo valor de `ProjectionType` sea `KEYS_ONLY` o `ALL`.

Claves de partición y claves de clasificación

Longitud de la clave de partición

La longitud mínima de un valor de clave de partición es de 1 byte. La longitud máxima es de 2048 bytes.

Valores de la clave de partición

No existe ningún límite práctico respecto al número de valores diferentes de clave de partición, ni para tablas ni para los índices secundarios.

Longitud de la clave de clasificación

La longitud mínima de un valor de clave de ordenación es de 1 byte. La longitud máxima es de 1024 bytes.

Valores de la clave de clasificación

En general, no existe ningún límite práctico respecto al número de valores diferentes de clave de ordenación por cada valor de clave de partición.

Hay una excepción en las tablas que utilizan índices secundarios. Una colección de elementos es el conjunto de elementos que tienen el mismo valor de atributo de clave de partición. En un índice secundario global, la colección de elementos es independiente de la tabla base (y puede tener un atributo de clave de partición diferente), pero en un índice secundario local la vista indexada se coubica en la misma partición que el elemento de la tabla y comparte el mismo atributo de clave de partición. Como resultado de esta ubicación, cuando una tabla tiene uno o más LSI, la colección de elementos no puede distribuirse en múltiples particiones.

En el caso de una tabla con uno o más LSI, las colecciones de elementos no pueden superar los 10 GB de tamaño. Se incluyen todos los elementos de la tabla base y todas las vistas LSI proyectadas que tengan el mismo valor del atributo de clave de partición. 10 GB es el tamaño máximo de una partición. Para obtener información más detallada, consulte Límite del tamaño de una colección de elementos.

Reglas de nomenclatura

Nombres de tabla y nombres del índice secundario

Los nombres de las tablas y de los índices secundarios deben tener 3 caracteres como mínimo y 255 como máximo. A continuación se muestran los caracteres permitidos:

- `A-Z`
- `a-z`
- `0-9`
- `_` (guion bajo)
- `-` (guion)
- `.` (punto)

Nombres de los atributos

En general, un nombre de atributo debe tener un carácter como mínimo y su tamaño máximo no debe superar los 64 KB.

A continuación se muestran las excepciones. Los siguientes nombres de atributo no pueden tener más de 255 caracteres:

- Los nombres de las claves de partición de índice secundario.
- Los nombres de las claves de ordenación de índices secundarios.
- Los nombres de los atributos proyectados especificados por el usuario (aplicables solo a índices secundarios locales). En una operación `CreateTable`, si especifica el valor `ProjectionType` para `INCLUDE`, los nombres de los atributos del parámetro `NonKeyAttributes` estarán sujetos a restricciones de longitud. Los tipos de proyecciones `KEYS_ONLY` y `ALL` no se ven afectados.

Estos nombres de atributos deben codificarse mediante UTF-8 y el tamaño total de cada nombre (después de la codificación) no puede superar los 255 bytes.

Tipos de datos

Cadena

La longitud de un valor de tipo String está limitada en función del tamaño de elemento máximo de 400 KB.

Los valores de tipo String son Unicode con codificación binaria UTF-8. Dado que UTF-8 es una codificación de ancho variable, DynamoDB determina los bytes UTF-8 de un valor de tipo String para determinar su longitud.

Número

Un valor de tipo Number puede tener hasta 38 dígitos de precisión y puede ser positivo, negativo o cero.

- Rango de valores positivos: de `1E-130` a `9.99999999999999999999999999999999999999E+125`
- Rango de valores negativos: de `-9.99999999999999999999999999999999999999E+125` a `-1E-130`

DynamoDB utiliza cadenas JSON para representar los datos de tipo Number en las solicitudes y las respuestas. Para obtener más información, consulte API de bajo nivel de DynamoDB.

Si la precisión del número es importante, debe pasar los números a DynamoDB utilizando cadenas que se conviertan a partir de un tipo de número.

Binario

La longitud de un valor de tipo Binary está limitada en función del tamaño de elemento máximo de 400 KB.

Las aplicaciones que utilizan atributos de tipo Binary deben codificar los datos en formato Base64 antes de enviárselos a DynamoDB. Al recibirlos, DynamoDB decodifica los datos y los convierte a matrices de bytes sin signo; a continuación, utiliza ese resultado como longitud del atributo.

Items

Tamaño del elemento

El tamaño máximo de un elemento en DynamoDB es de 400 KB, que incluye tanto la longitud en formato binario de los nombres de los atributos (longitud en UTF-8) y las longitudes de los valores de los atributos (también en formato binario). El nombre de los atributos se tiene en cuenta al calcular el límite de tamaño.

Por ejemplo, tomemos un elemento con dos atributos: un atributo denominado “shirt-color” cuyo valor es “R” y otro denominado “shirt-size” cuyo valor es “M”. El tamaño total de ese elemento es de 23 bytes.

Tamaño del elemento para las tablas con índices secundarios locales

Para cada índice secundario local de una tabla, existe un límite de 400 KB que se aplica al total de los siguientes tamaños:

- El tamaño de los datos de un elemento de la tabla.
- El tamaño de las entradas correspondientes (incluidos los valores clave y los atributos proyectados) en todos los índices secundarios locales.

Atributos

Pares de nombre-valor de los atributos por elemento

El tamaño acumulado de los atributos por elemento debe ajustarse al tamaño máximo de elemento de DynamoDB (400 KB).

Número de valores de una lista, un mapa o un conjunto

No existe ningún límite respecto al número de valores de una lista, un mapa o un conjunto, siempre y cuando el elemento que contenga los valores se ajuste al límite de tamaño de elemento de 400 KB.

Valores de los atributos

Se pueden emplear valores de atributo binarios o de cadena vacíos si el atributo no se utiliza como atributo de clave en una tabla o índice. En los conjuntos, listas y mapas, se admiten valores binarios y de cadena vacíos. El valor de un atributo no puede ser un conjunto vacío (conjunto de cadenas, conjunto de números o conjunto binario). Sin embargo, sí se admiten valores de tipo lista o mapa vacíos.

Profundidad de los atributos anidados

DynamoDB admite atributos anidados hasta un máximo de 32 niveles de profundidad.

Parámetros de expresión

Los parámetros de expresión incluyen `ProjectionExpression`, `ConditionExpression`, `UpdateExpression` y `FilterExpression`.

Longitudes

La longitud máxima de cualquier cadena de expresión es 4 KB. Por ejemplo, el tamaño de `ConditionExpression a=b` es de 3 bytes.

La longitud máxima de cualquier nombre de atributo de expresión individual o valor de atributo de expresión es de 255 bytes. Por ejemplo, en el caso de `#name`, es de 5 bytes, mientras que en el de `:val`, es de 4.

La longitud máxima de todas las variables de sustitución de una expresión es de 2 MB. Este valor representa la suma de las longitudes de todos los `ExpressionAttributeNames` y `ExpressionAttributeValues`.

Operadores y operandos

El número máximo de operadores o funciones que se admiten en una `UpdateExpression` es de 300. Por ejemplo, UpdateExpression `SET a = :val1 + :val2 + :val3` contiene dos operadores “+”.

La cantidad máxima de operandos del comparador `IN` es de 100.

Palabras reservadas

DynamoDB no impide utilizar nombres que entran en conflicto con las palabras reservadas. Para ver una lista completa, consulte Palabras reservadas en DynamoDB.)

Sin embargo, si utiliza una palabra reservada en un parámetro de expresión, también debe especificar `ExpressionAttributeNames`. Para obtener más información, consulte Nombres de atributos de expresión (Alias) en DynamoDB.

Transacciones de DynamoDB

Las operaciones de la API transaccionales de DynamoDB presentan las siguientes restricciones:

- Una transacción no puede contener más de 100 elementos únicos.
- Una transacción no puede contener más de 4 MB de datos.
- No se pueden aplicar dos acciones de una transacción al mismo elemento de la misma tabla. Por ejemplo, no se puede usar `ConditionCheck` y `Update` para el mismo elemento de una transacción.
- Una transacción no puede operar en tablas que se encuentren en más de una cuenta o región de AWS.
- Las operaciones transaccionales proporcionan garantías de atomicidad, uniformidad, aislamiento y durabilidad (ACID, por sus siglas en inglés) solo en la región de AWS en la que se crea la escritura originalmente. No se admiten las transacciones entre regiones en las tablas globales. Por ejemplo, supongamos que tiene una tabla global con réplicas en las regiones EE. UU. Este (Ohio) y EE. UU. Oeste (Oregón) y realiza una operación `TransactWriteItems` en la región EE. UU. Este (Norte de Virginia). En este caso, puede observar transacciones parcialmente completadas en la región EE. UU. Oeste (Oregón) a medida que se replican los cambios. Los cambios se replican en otras regiones solo cuando se han confirmado en la región de origen.

DynamoDB Streams

Lectores simultáneos de una partición en DynamoDB Streams

En el caso de las tablas de una sola región que no sean tablas globales, puede diseñar hasta dos procesos para leer desde la misma partición de DynamoDB Streams al mismo tiempo. Si excede este límite, puede producirse una limitación controlada de las solicitudes. En el caso de las tablas globales, le recomendamos que limite el número de lectores simultáneos a uno para evitar la limitación de solicitudes.

DynamoDB Accelerator (DAX)

Disponibilidad regional de AWS

Para obtener una lista de las regiones de AWS en las que DAX está disponible, consulte DynamoDB Accelerator (DAX) en la Referencia general de AWS.

Nodos

Un clúster de DAX consta exactamente de un nodo primario y entre cero y nueve nodos de réplica de lectura.

El número total de nodos (por cuenta de AWS) no puede superar los 50 en una misma región de AWS.

Grupos de parámetros

Puede crear hasta 20 grupos de parámetros de DAX por región.

Grupos de subredes

Puede crear hasta 50 grupos de subredes de DAX por región.

Dentro de un grupo de subredes, puede definir hasta 20 subredes.

> **Important**
>
> Un clúster de DAX admite un máximo de 500 tablas de DynamoDB. Una vez que supere las 500 tablas de DynamoDB, es posible que se produzca un deterioro de la disponibilidad y el rendimiento del clúster.

Restricciones específicas de la API

`CreateTable`/`UpdateTable`/`DeleteTable`/`PutResourcePolicy`/`DeleteResourcePolicy`

En general, pueden ejecutarse de forma simultánea hasta 500 solicitudes de CreateTable, UpdateTable, DeleteTable, PutResourcePolicy y DeleteResourcePolicy en cualquier combinación. Por consiguiente, el número total de tablas que se encuentren en estado `CREATING`, `UPDATING` o `DELETING` no puede ser mayor que 500.

Puede enviar hasta 2500 solicitudes por segundo de solicitudes de la API del plano de control (`CreateTable`, `DeleteTable`, `UpdateTable`, `PutResourcePolicy` y `DeleteResourcePolicy`) mutables en cualquier grupo de tablas. Sin embargo, las solicitudes `PutResourcePolicy` y `DeleteResourcePolicy` tienen límites individuales más bajos. Para obtener más información, consulte los siguientes detalles sobre las cuotas para `PutResourcePolicy` y `DeleteResourcePolicy`.

Las solicitudes `CreateTable` y `PutResourcePolicy` que incluyan una política basada en recursos contarán como dos solicitudes adicionales por cada KB de la política. Por ejemplo, una solicitud `CreateTable` o `PutResourcePolicy` con una política de tamaño de 5 KB contará como 11 solicitudes: 1 para la solicitud `CreateTable` y 10 para la política basada en recursos (2 x 5 KB). Del mismo modo, una política de tamaño de 20 KB contará como 41 solicitudes: 1 para la solicitud `CreateTable` y 40 para la política basada en recursos (2 x 20 KB).

`PutResourcePolicy`

Puede enviar hasta 25 solicitudes de la API `PutResourcePolicy` por segundo en un grupo de tablas. Tras solicitar una tabla individual correctamente, no se admiten nuevas solicitudes `PutResourcePolicy` durante los 15 segundos siguientes.

El tamaño máximo admitido para un documento de política basado en recursos es de 20 KB. DynamoDB cuenta los espacios en blanco al calcular el tamaño de una política según esta limitación.

`DeleteResourcePolicy`

Puede enviar hasta 50 solicitudes de la API `DeleteResourcePolicy` por segundo en un grupo de tablas. Tras realizar una solicitud `PutResourcePolicy` correcta para una tabla individual, no se admiten solicitudes `DeleteResourcePolicy` durante los 15 segundos siguientes.

BatchGetItem

En una sola operación `BatchGetItem` se puede recuperar un máximo de 100 elementos. El tamaño total de todos los elementos recuperados no puede ser mayor que 16 MB.

BatchWriteItem

Una misma operación `BatchWriteItem` puede contener hasta 25 solicitudes `PutItem` o `DeleteItem`. El tamaño total de todos los elementos escritos no puede ser mayor que 16 MB.

DescribeStream

Puede llamar a `DescribeStream` a una velocidad máxima de 10 veces por segundo.

DescribeTableReplicaAutoScaling

El método `DescribeTableReplicaAutoScaling` admite solo diez solicitudes por segundo.

DescribeLimits

Solo se debe llamar a `DescribeLimits` de forma periódica. Es de esperar que se produzcan errores de limitación controlada si se realiza la llamada más de una vez por minuto.

DescribeContributorInsights/ListContributorInsights/UpdateContributorInsights

Solo se debe llamar a `DescribeContributorInsights`, `ListContributorInsights` y `UpdateContributorInsights` de forma periódica. DynamoDB admite hasta cinco solicitudes por segundo para cada una de estas API.

DescribeTable/ListTables/GetResourcePolicy

Puede enviar hasta 2500 solicitudes por segundo de una combinación de solicitudes de la API del plano de control de solo lectura (`DescribeTable`, `ListTables` y `GetResourcePolicy`). La API `GetResourcePolicy` tiene un límite individual inferior de 100 solicitudes por segundo.

`DescribeTimeToLive`

La operación `DescribeTimeToLive` está limitada a 10 unidades de solicitud de lectura por segundo. Si supera este límite, DynamoDB devuelve un error `ThrottlingException`.

`Query`

El conjunto de resultados de una operación `Query` está limitado a 1 MB por llamada. Puede utilizar `LastEvaluatedKey` de la respuesta a la consulta para recuperar más resultados.

`Scan`

El conjunto de resultados de una operación `Scan` está limitado a 1 MB por llamada. Puede utilizar `LastEvaluatedKey` de la respuesta al examen para recuperar más resultados.

`UpdateKinesisStreamingDestination`

Al realizar operaciones `UpdateKinesisStreamingDestination`, puede establecer `ApproximateCreationDateTimePrecision` con un valor nuevo un máximo de 3 veces en un período de 24 horas.

`UpdateTableReplicaAutoScaling`

El método `UpdateTableReplicaAutoScaling` admite solo diez solicitudes por segundo.

`UpdateTableTimeToLive`

El método `UpdateTableTimeToLive` solo admite una solicitud para activar o desactivar `Time to Live (TTL)` por tabla especificada y por hora. Este cambio puede tardar hasta una hora en procesarse completamente. Cualquier llamada `UpdateTimeToLive` adicional para la misma tabla en esta hora de duración genera ValidationException.

Cifrado en reposo en DynamoDB

Puede cambiar entre una Clave propiedad de AWS, una Clave administrada de AWS y una clave administrada por el cliente hasta cuatro veces, cada 24 horas, por cada tabla, a partir del momento en el que se cree la tabla. En caso de no producirse ningún cambio en las últimas seis horas, se

permite un cambio adicional. Esto aumenta el número máximo de cambios a ocho al día (cuatro cambios en las primeras seis horas y uno en cada una de las seis horas posteriores durante un día).

Puede cambiar las claves de cifrado para usar una Clave propiedad de AWS con tanta frecuencia como sea preciso, incluso si se ha agotado la cuota anterior.

Estas son las cuotas a menos que solicite una cantidad mayor. Para solicitar un aumento de la cuota de servicio, veahttps://aws.amazon.com/support.

Referencia de la API de DynamoDB

La Referencia de la API de Amazon DynamoDB contiene una lista completa de las operaciones admitidas por:

- DynamoDB
- DynamoDB Streams
- DynamoDB Accelerator (DAX)

Solución de problemas de Amazon DynamoDB

Los siguientes temas le proporcionan consejos para solucionar errores y problemas que pueda encontrar al utilizar Amazon DynamoDB. Si se encuentra con un problema que no se muestra aquí, puede utilizar el botón de feedback de esta página para notificarlo.

Para obtener más consejos sobre la resolución de problemas y respuestas a preguntas comunes de soporte, visite el Centro de conocimientos de AWS.

Temas

- Solución de problemas de errores del servidor internos en Amazon DynamoDB
- Solución de problemas de latencia en Amazon DynamoDB
- Solución de problemas de limitación en Amazon DynamoDB

Solución de problemas de errores del servidor internos en Amazon DynamoDB

En DynamoDB, los errores del servidor internos (errores 500) indican que el servicio no puede atender la solicitud. Estos errores se pueden producir por varios motivos, como problemas transitorios de red en la flota, problemas de infraestructura, problemas relacionados con los nodos de almacenamiento, etc.

Es posible que se produzcan algunos errores del servidor internos durante el ciclo de vida de la tabla de DynamoDB. Esto es normal debido a la naturaleza distribuida del servicio y, por lo general, no debería ser motivo de preocupación. DynamoDB repara y soluciona automáticamente cualquier problema transitorio del servicio en tiempo real, sin que sea necesaria su intervención. Sin embargo, si observa un número elevado y constante de errores del servidor internos en las solicitudes a la tabla (como se ve en la métrica the section called “SystemErrors”), debería investigar con más detalle.

Temas

- Investigación de los errores del servidor internos
- Minimización del impacto de los errores del servidor internos
- Mejora del conocimiento operativo

Investigación de los errores del servidor internos

Si encuentra errores del servidor internos en la tabla de DynamoDB, considere estas opciones:

1. Consulte el panel de AWS Health.

   Para identificar el problema, el primer paso es revisar el Panel de estado del servicio de AWS y el panel de estado de la cuenta de AWS. Estos paneles proporcionan información valiosa sobre cualquier problema que afecte a todo el servicio, las tablas afectadas, los problemas actuales y la causa raíz una vez resuelto el problema.

   La revisión de los detalles de estos paneles le permitirá comprender mejor el estado actual de los Servicios de AWS que está utilizando y cualquier problema potencial que afecte a la cuenta. Esta información puede ayudarle a determinar los próximos pasos para abordar el problema y minimizar cualquier interrupción en las operaciones.

2. Contacte con Soporte.

   Si observa errores prolongados y persistentes en las solicitudes, es posible que se trate de un problema con el servicio. Como regla general, si observa una tasa de errores general del 1 % o más en los últimos 15 minutos, es el momento adecuado para plantear el problema al equipo de AWS Support. Consulte el Acuerdo de nivel de servicio de DynamoDB para obtener más información.

   Al abrir un caso con el equipo de AWS Support, proporcione los siguientes detalles para ayudar a agilizar el proceso de solución de problemas:

   - DDB afectado; tablas o índices secundarios
   - Periodo temporal en el que se observaron los errores
   - ID de solicitud de DynamoDB, como `4KBNVRGD25RG1KE09UT4V3FQDJVV4KQNSO5AEMVJF66Q9ASUAAJG`, que puede encontrar en los registros de la aplicación.

   La inclusión de estos detalles en el caso de soporte ayudará al equipo de AWS a entender el problema y a resolverlo más rápido. Si no tiene los ID de la solicitud, debe registrar el caso con el resto de detalles disponibles.

Minimización del impacto de los errores del servidor internos

Si se producen errores del servidor internos cuando se use DynamoDB, minimice el impacto de los mismos en la aplicación, tenga en cuenta las siguientes prácticas recomendadas:

- Uso de retrasos y reintentos: los comportamientos predeterminados del SDK de DynamoDB están diseñados para encontrar el equilibrio adecuado para la mayoría de las aplicaciones en términos de estrategia de retrasos y reintentos. Sin embargo, puede ajustar esta configuración en función de la tolerancia de la aplicación al tiempo de inactividad y a los requisitos de rendimiento. Obtenga más información sobre los retrasos y los reintentos para saber cómo puede ajustar esta configuración de reintento.
- Uso de lecturas coherentes posteriores: si la aplicación no requiere lecturas altamente coherentes, considere la posibilidad de utilizar lecturas coherentes posteriores. Estas lecturas cuestan menos y es menos probable que se produzcan problemas transitorios debido a errores del servidor internos, ya que se realizarían desde cualquiera de los nodos de almacenamiento disponibles. Para obtener más información, consulte Coherencia de lectura de DynamoDB.

Mejora del conocimiento operativo

El mantenimiento de una alta disponibilidad y fiabilidad de las aplicaciones es crucial en el panorama digital actual. Un aspecto clave de esto es la supervisión proactiva de los errores del servidor internos (ISE) en las tablas y los índices secundarios globales (GSI) de DynamoDB. Al crear alarmas de CloudWatch para supervisar estos errores, puede obtener un mejor conocimiento operativo y recibir alertas sobre posibles problemas antes de que afecten a los usuarios finales. Este enfoque se alinea con el pilar de excelencia operativa del marco de AWS Well-Architected, lo que garantiza que la carga de trabajo de DynamoDB esté optimizada en términos de rendimiento, seguridad y fiabilidad.

Creación de alarmas de CloudWatch

Debe configurar las alarmas de CloudWatch en las tablas de DynamoDB para recibir notificaciones de un número elevado y constante de errores internos del servidor, en lugar de observar las métricas manualmente. Esto se relaciona con el pilar de excelencia operativa del marco Well-Architected para cualquier carga de trabajo en AWS. Consulte Uso del enfoque Well-Architected de DynamoDB para optimizar su carga de trabajo de DynamoDB para obtener más información sobre cómo diseñar correctamente las tablas de DynamoDB.

Estas alarmas utilizan métricas personalizadas para calcular el porcentaje de solicitudes erróneas en un periodo de 5 minutos. La práctica recomendada es configurar la alarma para que entre en el

estado ALARM cuando 3 puntos de datos consecutivos superen el umbral del 1 %, lo que significa que, en general, el 1 % de las solicitudes producen un error en un periodo de 15 minutos.

El siguiente ejemplo es una plantilla de CloudFormation que puede ayudarle a crear alarmas de CloudWatch en la tabla y GSI en la tabla.

```
AWSTemplateFormatVersion: "2010-09-09"
Description: Sample template for monitoring DynamoDB
Parameters:
 DynamoDBProvisionedTableName:
    Description: Name of DynamoDB Provisioned Table to create
    Type: String
    MinLength: 3
    MaxLength: 255
    ConstraintDescription : https://docs.aws.amazon.com/amazondynamodb/latest/
developerguide/Limits.html#limits-naming-rules
  DynamoDBSNSEmail:
    Description : Email Address subscribed to newly created SNS Topic
    Type: String
    AllowedPattern: "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+$"
    MinLength: 1
    MaxLength: 255

Resources:
  DynamoDBMonitoringSNSTopic:
    Type: AWS::SNS::Topic
    Properties:
      DisplayName: DynamoDB Monitoring SNS Topic
      Subscription:
        - Endpoint: !Ref DynamoDBSNSEmail
          Protocol: email
      TopicName: dynamodb-monitoring

  DynamoDBTableSystemErrorAlarm:
    Type: 'AWS::CloudWatch::Alarm'
    Properties:
      AlarmName: 'DynamoDBTableSystemErrorAlarm'
      AlarmDescription: 'Alarm when system errors exceed 1% of total number of requests
 for 15 minutes'
      AlarmActions:
        - !Ref DynamoDBMonitoringSNSTopic
      Metrics:
        - Id: 'e1'
```

```
          Expression: 'm1/(m1+m2+m3)'
          Label: SystemErrorsOverTotalRequests
        - Id: 'm1'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'SystemErrors'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
        - Id: 'm2'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedReadCapacityUnits'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
        - Id: 'm3'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedWriteCapacityUnits'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
      EvaluationPeriods: 3
      Threshold: 1.0
      ComparisonOperator: 'GreaterThanThreshold'
  DynamoDBGSISystemErrorAlarm:
    Type: 'AWS::CloudWatch::Alarm'
    Properties:
```

```
      AlarmName: 'DynamoDBGSISystemErrorAlarm'
      AlarmDescription: 'Alarm when GSI system errors exceed 2% of total number of
 requests for 15 minutes'
      AlarmActions:
        - !Ref DynamoDBMonitoringSNSTopic
      Metrics:
        - Id: 'e1'
          Expression: 'm1/(m1+m2+m3)'
          Label: GSISystemErrorsOverTotalRequests
        - Id: 'm1'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'SystemErrors'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
                - Name: 'GlobalSecondaryIndexName'
                  Value: !Join [ '-', [!Ref DynamoDBProvisionedTableName, 'gsi1'] ]
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
        - Id: 'm2'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedReadCapacityUnits'
              Dimensions:
                - Name: 'TableName'
                  Value: !Ref DynamoDBProvisionedTableName
                - Name: 'GlobalSecondaryIndexName'
                  Value: !Join [ '-', [!Ref DynamoDBProvisionedTableName, 'gsi1'] ]
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
          ReturnData: False
        - Id: 'm3'
          MetricStat:
            Metric:
              Namespace: 'AWS/DynamoDB'
              MetricName: 'ConsumedWriteCapacityUnits'
              Dimensions:
                - Name: 'TableName'
```

```
                Value: !Ref DynamoDBProvisionedTableName
              - Name: 'GlobalSecondaryIndexName'
                Value: !Join [ '-', [!Ref DynamoDBProvisionedTableName, 'gsi1'] ]
            Period: 300
            Stat: 'SampleCount'
            Unit: 'Count'
        ReturnData: False
    EvaluationPeriods: 3
    Threshold: 1.0
    ComparisonOperator: 'GreaterThanThreshold'
```

Solución de problemas de latencia en Amazon DynamoDB

Si la carga de trabajo parece experimentar una alta latencia, puede analizar la métrica `SuccessfulRequestLatency` de CloudWatch y comprobar la latencia media y la latencia mediana a través de métricas de percentil (p50) para ver si está relacionada con DynamoDB. Es normal que haya cierta variabilidad en `SuccessfulRequestLatency` notificado y los picos ocasionales (en concreto en la estadística de `Maximum` y los percentiles) no deben ser motivo de preocupación. No obstante, si la estadística de `Average` o p50 (mediana) muestra un fuerte aumento y persiste, debe consultar el panel de estado del servicio de AWS y el panel de estado personal para obtener más información. Algunas causas posibles son el tamaño del elemento de la tabla (un elemento de 1 KB y otro de 400 KB variarán en latencia) o el tamaño de la consulta (10 elementos frente a 100 elementos).

Las métricas de percentil (p99, p90, etc.) pueden ayudarlo a comprender mejor la distribución de latencia. Por ejemplo:

- p50 (mediana) muestra la latencia típica de la carga de trabajo.
- p90 muestra que el 90 % de las solicitudes son más rápidas que este valor.
- p99 ayuda a identificar la latencia en el peor de los casos, que afecta al 1 % de las solicitudes.

Los valores altos de p99 con valores normales de p50 podrían indicar problemas esporádicos que afectan una pequeña parte de las solicitudes, mientras que los valores elevados de p50 de forma coherente podrían sugerir cierto deterioro del rendimiento.

> **Note**
>
> Para analizar valores de percentiles personalizados (como p99,9), puede introducir manualmente el percentil deseado (por ejemplo, p99,9) en el campo de estadísticas de métricas de CloudWatch. Esto le permite evaluar las distribuciones de latencia más allá de los percentiles predeterminados que aparecen en la lista desplegable.

Se espera cierta variación en las métricas de latencia, en concreto en los percentiles más altos, y puede verse como resultado de las operaciones en segundo plano impulsadas por DynamoDB que ayudan a mantener una alta disponibilidad y durabilidad de los datos almacenados en tablas de DynamoDB o problemas transitorios de infraestructura.

Si es necesario, considere la posibilidad de abrir un caso de soporte con AWS Support y siga evaluando las opciones de emergencia disponibles para su aplicación (como la evacuación de una región si tiene una arquitectura multirregión) de acuerdo con sus manuales de procedimientos. Debes registrar los ID de solicitud de las solicitudes lentas para proporcionarlos a AWS Support cuando cree un caso de asistencia.

La métrica `SuccessfulRequestLatency` solo mide la latencia interna del servicio DynamoDB; no se incluyen la actividad del cliente ni los tiempos de ida y vuelta de la red. Para obtener más información sobre la latencia global de las llamadas de su cliente al servicio DynamoDB, puede activar el registro de métricas de latencia en su SDK de AWS.

> **Note**
>
> Para la mayoría de las operaciones singleton (operaciones que se aplican a un único elemento mediante la especificación completa del valor de la clave principal), DynamoDB proporciona `Average SuccessfulRequestLatency` milisegundos de un solo dígito. Este valor no incluye la sobrecarga de transporte para el código de llamada que accede al punto de conexión de DynamoDB. En el caso de las operaciones con datos de varios elementos, la latencia variará en función de factores como el tamaño del conjunto de resultados, la complejidad de las estructuras de datos devueltas y las expresiones de condición y de filtro aplicadas. Para operaciones repetidas de varios elementos al mismo conjunto de datos con los mismos parámetros, DynamoDB proporcionará `Average SuccessfulRequestLatency` de alta coherencia.

Considere una o más de las siguientes estrategias para reducir la latencia:

- Reutilizar conexiones: las solicitudes de DynamoDB se realizan a través de una sesión autenticada que es HTTPS de forma predeterminada. Iniciar la conexión requiere varios procesos de ida y vuelta y lleva tiempo, por lo que la latencia de la primera solicitud es superior a la de las siguientes solicitudes que reutilizan la conexión. Las solicitudes a través de una conexión ya inicializada ofrecen la baja latencia coherente de DynamoDB. Para evitar la latencia que supone establecer nuevas conexiones, es posible que desee implementar un mecanismo de "mantenimiento de conexión" mediante el envío de una solicitud `GetItem` cada 30 segundos si no se realizan otras solicitudes.
- Usar lecturas coherentes posteriores: si su aplicación no requiere lecturas altamente coherentes, considere la posibilidad de utilizar las lecturas coherentes posteriores predeterminadas. Las lecturas coherentes posteriores tienen un costo menor y pueden provenir de varias zonas de disponibilidad, lo que permite seleccionar una zona de disponibilidad ubicada cerca del solicitante y reducir la latencia. Para obtener más información, consulte Coherencia de lectura de DynamoDB.
- Implementar cobertura de solicitudes: para requisitos de latencia p99 muy bajos, considere implementar cobertura de solicitudes. Con la cobertura de solicitudes, si la solicitud inicial no recibe una respuesta lo suficientemente rápido, envíe una segunda solicitud equivalente y deje que compitan. La primera respuesta gana. Esto mejora la latencia de cola a costa de algunas solicitudes adicionales. Puede decidir cuánto tiempo esperar antes de enviar la segunda solicitud. La cobertura es más fácil para las lecturas. Para las escrituras, utilice el orden basado en marcas temporales para garantizar que las solicitudes cubiertas se traten como si ocurrieran en el momento del primer intento, evitando actualizaciones desordenadas. Este enfoque se ha analizado en Escrituras con marca temporal para cobertura de escritura en Amazon DynamoDB.
- Ajustar el tiempo de espera de la solicitud y el comportamiento de reintentos: la ruta desde el cliente a DynamoDB atraviesa muchos componentes, cada uno diseñado teniendo en cuenta la redundancia. Tenga en cuenta los siguientes aspectos:
  - Resiliencia de la red
  - Tiempos de espera de los paquetes TCP
  - Arquitectura distribuida de DynamoDB

  Los comportamientos del SDK predeterminados están optimizados para la mayoría de las aplicaciones. No obstante, puede implementar una estrategia de respuesta rápida a los errores y ajustar la configuración del tiempo de espera. Las solicitudes que tardan mucho más de lo normal tienen menos probabilidades de realizarse correctamente. Al responder rápido a los errores y

reintentarlo, es posible que lo consiga rápidamente a través de una ruta diferente. Esto es similar a la cobertura de solicitudes, pero finaliza la primera solicitud en lugar de permitir que continúe.

Evite establecer valores de tiempo de espera demasiado bajos. Los tiempos de espera demasiado bajos pueden provocar problemas de disponibilidad inducidos por el cliente. Por ejemplo, un tiempo de espera de socket de 50 milisegundos podría provocar errores de conexión durante picos de latencia de la red, por ejemplo, cuando se acercan los límites de ancho de banda de la instancia de Amazon EC2 para el tráfico de flujo único. Valore detenidamente las ventajas de los tiempos de espera más bajos frente a los riesgos potenciales para la disponibilidad de la aplicación y priorice la cobertura a los tiempos de espera cortos.

Para obtener un análisis útil de este tema, consulte Tuning AWS Java SDK HTTP request settings for latency-aware Amazon DynamoDB applications.

- Reducir la distancia entre el cliente y el punto de conexión de DynamoDB: si tiene usuarios dispersos por todo el mundo, considere la posibilidad de utilizar Tablas globales: multiactivas, replicación en varias regiones. Con las tablas globales, puede replicar la tabla en las regiones de AWS especificadas donde desee que esté disponible. Puede colocar una copia de los datos más cerca del usuario final para reducir la latencia de red durante las operaciones de lectura y escritura. Para obtener más información sobre cómo utilizar las tablas globales de DynamoDB de forma efectiva, consulte Uso de tablas globales de Amazon DynamoDB en Recomendaciones de AWS.
- Utilizar el almacenamiento en caché: si tiene mucho tráfico de lectura, considere la posibilidad de utilizar uno de estos servicios de almacenamiento en caché:
  - Acelerador de DynamoDB (DAX) es una caché en memoria altamente disponible y completamente administrada para DynamoDB que multiplica el rendimiento hasta por diez (de milisegundos a microsegundos) incluso con millones de solicitudes por segundo. Para obtener más información sobre DAX, consulte Aceleración en memoria con DynamoDB Accelerator (DAX):
  - Amazon ElastiCache: un servicio de almacenamiento en caché en memoria totalmente administrado que se puede integrar con DynamoDB para mejorar el rendimiento de lectura mediante el patrón de reserva de caché. Para obtener más información, consulte Integrating Amazon DynamoDB and Amazon ElastiCache by using read-through caching en Recomendaciones de AWS.

Solución de problemas de limitación en Amazon DynamoDB

DynamoDB implementa la limitación con dos propósitos principales: mantener el rendimiento general del servicio y el control de los costos. La limitación sirve como una medida de protección intencionada que evita la degradación del rendimiento cuando las tasas de consumo superan la capacidad o como un mecanismo de control de costos cuando se alcanzan los límites máximos de rendimiento o cuota de servicio. Cuando se produce una limitación, DynamoDB devuelve excepciones específicas con información detallada sobre por qué se ha limitado la solicitud y qué recurso se ha visto afectado. Cada motivo de la limitación corresponde a métricas específicas de CloudWatch que proporcionan información adicional sobre la frecuencia y los patrones de los eventos de limitación.

El siguiente diagrama ilustra los cuatro escenarios principales en los que DynamoDB implementa la limitación de protección:

1. Se ha superado el rendimiento del rango clave (en ambos modos):

   El consumo dirigido a particiones específicas supera los límites de rendimiento internos por partición.

2. Se ha superado el rendimiento aprovisionado (en modo aprovisionado):

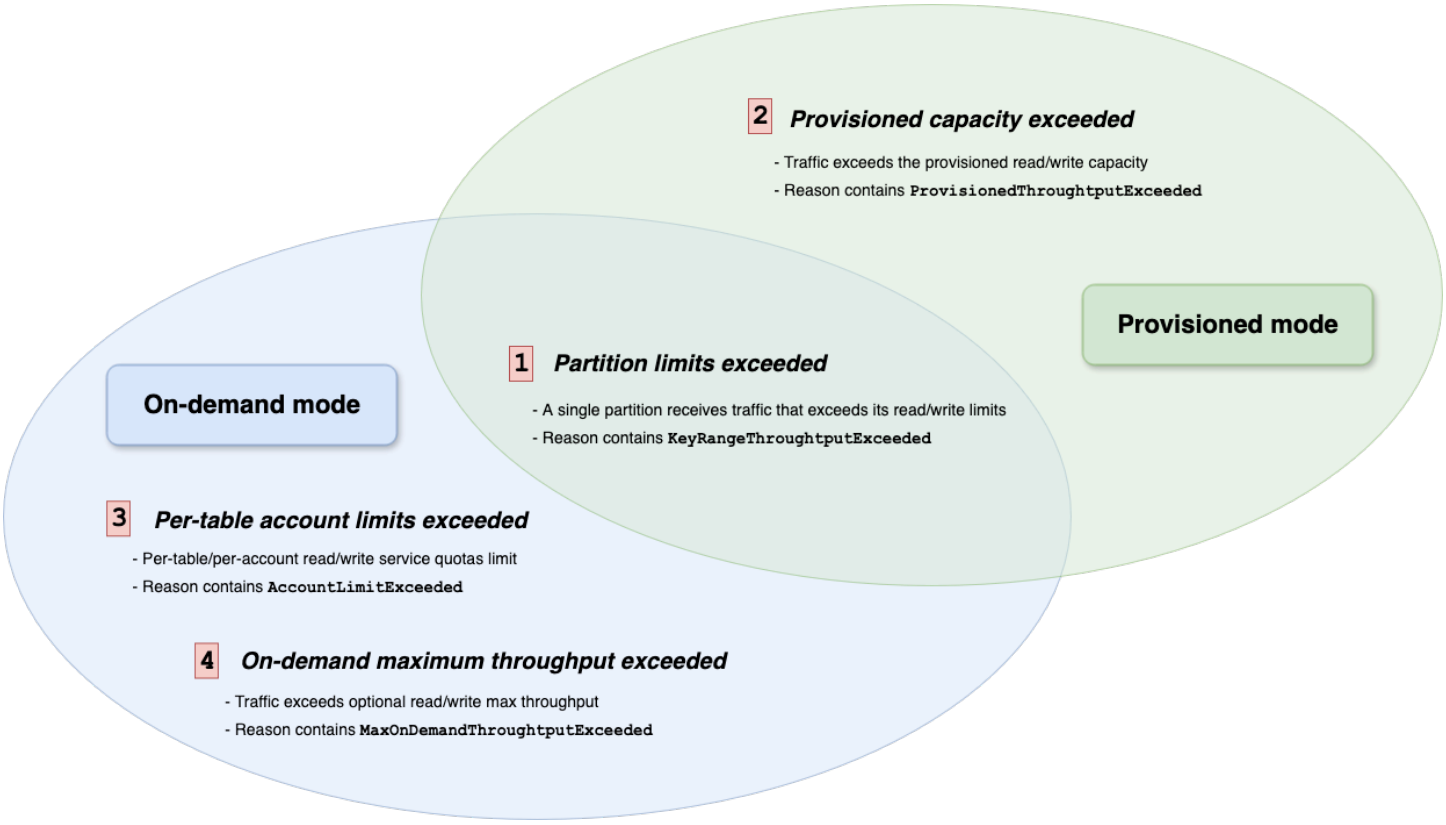
   El consumo supera las unidades de capacidad aprovisionadas (lectura o escritura) configuradas para una tabla o un índice secundario global (GSI).

3. Se han superado las cuotas de servicio por cuenta (en modo bajo demanda):

   El consumo hace que una tabla o un GSI supere las cuotas de servicio por tabla por cuenta en cuanto al rendimiento de lectura/escritura en la región de AWS actual. Estas cuotas sirven como protecciones y se pueden aumentar.

4. Se ha superado el rendimiento máximo bajo demanda (en modo bajo demanda):

   El consumo supera los límites máximos de rendimiento configurados para una tabla o GSI. Estos son límites que se configuran específicamente con fines de control de costos.



Esta guía está organizada para ayudarle a comprender y trabajar con la limitación en DynamoDB. En primer lugar, le ayudamos a identificar el tipo específico de limitación que afecta a la carga de trabajo mediante un marco de diagnóstico.

A continuación, la sección de la guía de resolución ofrece una guía específica para cada escenario de limitación, incluidas las métricas de CloudWatch para supervisar la detección y el análisis, y los pasos recomendados para la optimización. Si sigue este enfoque estructurado, podrá diagnosticar mejor la causa raíz de la limitación e implementar la solución adecuada para garantizar que las tablas de DynamoDB funcionen de manera eficiente.

Para empezar, siga the section called “Diagnóstico de limitación” para obtener información sobre cómo identificar qué tipo de limitación afecta a la carga de trabajo e implementar la estrategia de resolución recomendada.

Temas

- Diagnóstico de limitación
- Guía de resolución de limitación de DynamoDB
- Comprensión de la limitación de escritura del Índice Secundario Global (GSI) y la contrapresión en DynamoDB

- Métricas de limitación de CloudWatch

Diagnóstico de limitación

Cuando la aplicación experimenta limitación, DynamoDB proporciona información detallada sobre las excepciones y métricas específicas de CloudWatch para ayudarle a diagnosticar estos eventos.

En esta sección, se presenta un enfoque sistemático para entender los eventos de limitación en las aplicaciones de DynamoDB. Muestra cómo interpretar las excepciones de limitación, correlacionarlas con las métricas de CloudWatch para obtener información más detallada y comprender qué cambios reducirían la limitación en las aplicaciones de DynamoDB.

Comprensión de las excepciones de limitación

Cuando DynamoDB limita una solicitud, devuelve excepciones específicas con información de diagnóstico detallada. Por ejemplo, en Java, estas incluyen `ProvisionedThroughputExceededException`, `RequestLimitExceeded` o `ThrottlingException`.

Cada excepción incluye `ThrottlingReasons`, una recopilación de `ThrottlingReason` individuales que contiene dos campos clave para ayudarle a identificar y comprender la limitación:

- Un motivo: un campo concatenado que sigue el formato `<ResourceType><OperationType><LimitType>`
- Un ARN de recurso: el nombre de recurso de Amazon (ARN) de la tabla o índice afectado

El campo motivo sigue un patrón coherente que le ayuda a entender exactamente lo que está sucediendo:

- ResourceType (qué se está limitando): `Table` o `Index`
- OperationType (qué tipo de operación): `Read` o `Write`
- LimitType (motivo por el que se produjo la limitación):
  - `KeyRangeThroughputExceeded`: esto se produce cuando una partición específica que respalda la tabla o el índice ha consumido una capacidad de lectura o escritura que supera los límites de rendimiento internos por partición.

- `ProvisionedThroughputExceeded`: esto se produce en una tabla aprovisionada o en un índice secundario global cuando la tasa de consumo de lectura o escritura ha superado la cantidad aprovisionada.
- `AccountLimitExceeded`: esto se produce en una tabla o índice bajo demanda cuando la tasa de consumo de lectura o escritura supera la tasa de consumo máxima para una tabla y sus índices, establecida por cuenta. Puede solicitar un aumento de esta cuota.
- `MaxOnDemandThroughputExceeded`: esto se produce en una tabla o índice bajo demanda cuando la tasa de consumo de lectura o escritura ha superado la tasa de consumo máxima proporcionada por el usuario configurada para la tabla o el índice. Puede aumentar este valor usted mismo a cualquier valor hasta el límite de la cuenta o establecerlo en -1 para indicar que el usuario no ha proporcionado límite.

El ARN del recurso identifica exactamente qué tabla o índice se está limitando:

- Para tablas: `arn:aws:dynamodb:[region]:[account-id]:table/[table-name]`
- Para índices: `arn:aws:dynamodb:[region]:[account-id]:table/[table-name]/index/[index-name]`

Ejemplos de motivos de limitación total:

- `TableReadProvisionedThroughputExceeded`
- `IndexWriteAccountLimitExceeded`

Esto ayuda a identificar exactamente qué recurso se está limitando, qué tipo de operación lo provocó y por qué se produjo la limitación.

Excepciones de ejemplo

Ejemplo 1: se ha superado la capacidad aprovisionada en un GSI

```
{
    "ThrottlingReasons": [
        {
            "reason": "IndexWriteProvisionedThroughputExceeded",
            "resource": "arn:aws:dynamodb:us-west-2:123456789012:table/CustomerOrders/
index/OrderDateIndex"
        }
```

```
    ],
    "awsErrorDetails": {
        "errorCode": "ProvisionedThroughputExceeded",
        "errorMessage": "The level of configured provisioned throughput for the index
 was exceeded",
        "serviceName": "DynamoDB",
        "sdkHttpResponse": {
            "statusText": "Bad Request",
            "statusCode": 400
        }
    }
}
```

En este ejemplo, la aplicación recibe una `ProvisionedThroughputExceededException` con el motivo `IndexWriteProvisionedThroughputExceeded`. Las escrituras en el `OrderDateIndex` se están limitando porque el consumo de escritura ha superado la capacidad de escritura aprovisionada configurada para el GSI.

Ejemplo 2: Se ha superado el rendimiento máximo bajo demanda

```
{
    "ThrottlingReasons": [
        {
            "reason": "TableReadMaxOnDemandThroughputExceeded",
            "resource": "arn:aws:dynamodb:us-east-1:123456789012:table/UserSessions"
        }
    ],
    "awsErrorDetails": {
        "errorMessage": "Throughput exceeds the maximum OnDemandThroughput configured
 on table or index",
        "errorCode": "ThrottlingException",
        "serviceName": "DynamoDB",
        "sdkHttpResponse": {
            "statusText": "Bad Request",
            "statusCode": 400
        }
    }
}
```

En este ejemplo, las lecturas de la tabla `UserSessions` se están limitando porque superan el límite máximo de rendimiento bajo demanda configurado en la tabla.

Marco de diagnóstico de limitación de DynamoDB

Cuando la aplicación encuentre limitaciones, siga estos pasos para diagnosticar y resolver el problema.

Paso 1: analizar los detalles de **`ThrottlingReason`**

1. Compruebe el campo de motivo para identificar el motivo específico de la limitación. El motivo detalla el tipo de recurso limitado (tabla o índice), el tipo de operación que provoca el evento de limitación (lectura o escritura) y el tipo de límite que se ha superado (partición, rendimiento aprovisionado, límite de cuenta).
2. Compruebe el campo resourceArn para identificar qué recurso (tabla o GSI) se está limitando.
3. Utilice esta información combinada para comprender todo el contexto del problema de la limitación.

   Por ejemplo, considere este escenario en el que recibe la siguiente excepción `ProvisionedThroughputExceededException` con un motivo de limitación `TableWriteKeyRangeThroughputExceeded`. El resourceARN afectado es `arn:aws:dynamodb:us-west-2:123456789012:table/CustomerOrders`.

   Esta combinación le informa de que las operaciones de escritura en la tabla de `CustomerOrders` se están limitando. La limitación se produce por partición (no por tabla, que se mostraría como `TableWriteProvisionedThroughputExceeded`). La causa principal es que ha superado la capacidad de rendimiento máxima para un valor o rango de clave de partición específico, lo que indica que hay un problema de partición caliente.

   Comprender esta relación entre los elementos de excepción le ayudará a implementar la estrategia de mitigación adecuada; en este caso, abordar la partición caliente en lugar de aumentar la capacidad aprovisionada general de la tabla.

Paso 2: identificar y analizar las métricas de CloudWatch relacionadas

1. Identifique las métricas: cada motivo de limitación en DynamoDB corresponde directamente a métricas específicas de CloudWatch que puede supervisar para realizar un seguimiento y analizar los eventos de limitación. Puede derivar sistemáticamente los nombres de las métricas de CloudWatch adecuados a partir del motivo de la limitación.
2. Haga coincidir el motivo de la limitación con las métricas de CloudWatch correspondientes mediante esta tabla de referencia:

Referencia completa sobre los motivos de la limitación y las métricas de CloudWatch

Categoría	Motivo de la limitación	Métricas de CloudWatch principales
Se ha superado la capacidad aprovisionada	TableReadProvisionedThroughputExceeded	ReadProvisionedThroughputThrottleEvents
	TableWriteProvisionedThroughputExceeded	WriteProvisionedThroughputThrottleEvents
	IndexReadProvisionedThroughputExceeded	ReadProvisionedThroughputThrottleEvents (GSI)
	IndexWriteProvisionedThroughputExceeded	WriteProvisionedThroughputThrottleEvents (GSI)
Se han superado los límites de partición	TableReadKeyRangeThroughputExceeded	ReadKeyRangeThroughputThrottleEvents
	TableWriteKeyRangeThroughputExceeded	WriteKeyRangeThroughputThrottleEvents
	IndexReadKeyRangeThroughputExceeded	ReadKeyRangeThroughputThrottleEvents (GSI)
	IndexWriteKeyRangeThroughputExceeded	WriteKeyRangeThroughputThrottleEvents (GSI)

Categoría	Motivo de la limitación	Métricas de CloudWatch principales
Se ha superado el máximo bajo demanda	TableReadMaxOnDemandThroughputExceeded	ReadMaxOnDemandThroughputThrottleEvents
	TableWriteMaxOnDemandThroughputExceeded	WriteMaxOnDemandThroughputThrottleEvents
	IndexReadMaxOnDemandThroughputExceeded	ReadMaxOnDemandThroughputThrottleEvents (GSI)
	IndexWriteMaxOnDemandThroughputExceeded	WriteMaxOnDemandThroughputThrottleEvents (GSI)
Se han superado los límites de la cuenta	TableReadAccountLimitExceeded	ReadAccountLimitThrottleEvents
	TableWriteAccountLimitExceeded	WriteAccountLimitThrottleEvents
	IndexReadAccountLimitExceeded	ReadAccountLimitThrottleEvents (GSIs)
	IndexWriteAccountLimitExceeded	WriteAccountLimitThrottleEvents (GSIs)

Por ejemplo, si ha recibido `IndexWriteProvisionedThroughputExceeded`, como mínimo, debe supervisar la métrica de CloudWatch `WriteProvisionedThroughputThrottleEvents` para el índice específico identificado en el `ResourceArn`.

3. Supervise estas métricas en CloudWatch para comprender la frecuencia y el momento de los eventos de limitación, diferenciar entre la limitación de lectura y escritura, identificar patrones de tiempo cuando la limitación aumenta y realizar un seguimiento de las tendencias de utilización de la capacidad.

   DynamoDB publica métricas detalladas para cada tabla e índice secundario global. Las métricas (`ReadThrottleEvents`, `WriteThrottleEvents` y `ThrottledRequests`) agregan todos los eventos de limitación de la tabla y sus índices.

Paso 3: identificar las claves limitadas y las tasas de acceso altas mediante CloudWatch Contributor Insights (para limitación relacionada con particiones)

Si identificó problemas relacionados con las particiones en el paso 1 (por ejemplo, errores `KeyRangeThroughputExceeded`), CloudWatch Contributor Insights para DynamoDB puede ayudarlo a diagnosticar qué claves específicas generan tráfico y experimentan eventos de limitación en la tabla o índice.

1. Habilite CloudWatch Contributor Insights para la tabla o índice limitado en función del `ResourceARN`.

   Puede elegir el modo de Claves limitadas para centrarse exclusivamente en las claves más limitadas. Este modo es ideal para la supervisión continua, ya que solo procesa los eventos cuando se produce una limitación. Como alternativa, el modo Claves a las que se ha accedido y limitadas le ayuda a buscar patrones en las claves a las que más se accede.
2. Analice los informes para identificar los patrones problemáticos. Busque claves con tasas de acceso o de limitación desproporcionadamente altas y correlacione los patrones de limitación y tráfico. Puede crear paneles integrados que combinen los gráficos de Contributor Insights y las métricas de DynamoDB CloudWatch.

Para obtener más información sobre cómo habilitar y usar CloudWatch Contributor Insights, consulte Uso de CloudWatch Contributor Insights para DynamoDB.

Paso 4: determinar la solución adecuada

Tras diagnosticar la causa específica de la limitación, implemente la solución recomendada en función del contexto específico. La solución adecuada depende de varios factores, incluyendo el escenario de limitación, el modo de capacidad de la tabla, las decisiones de diseño de la tabla y las

claves, los patrones de acceso y la eficiencia de las consultas, la configuración de índices globales y secundarios, y la arquitectura general del sistema y los puntos de integración.

Para obtener soluciones detalladas que aborden los escenarios de limitación específicos, consulte la sección the section called “Guía de resolución”. Este recurso proporciona estrategias de corrección específicas personalizadas según el motivo concreto de la limitación y la configuración del modo de capacidad.

Paso 5: supervisar el progreso

1. Realice un seguimiento de las métricas de CloudWatch que correspondan al escenario de limitación.
2. Compruebe que las estrategias de mitigación son efectivas observando una disminución en los eventos de limitación.

Guía de resolución de limitación de DynamoDB

En esta sección, se proporciona una guía de resolución específica para cada motivo de limitación que DynamoDB pueda devolver. Cada entrada incluye enfoques de resolución sugeridos basados en las prácticas recomendadas y las métricas de CloudWatch correspondientes para supervisar.

DynamoDB implementa 16 motivos distintos de limitación en cuatro categorías principales. Utilice los motivos de limitación de la excepción de la aplicación para ir directamente a la guía correspondiente.

Se ha superado el rendimiento del rango clave (particiones calientes)

Estos motivos de limitación se producen cuando las particiones individuales superan los límites de rendimiento, lo que afecta al modo aprovisionado y al modo bajo demanda:

- TableReadKeyRangeThroughputExceeded
- TableWriteKeyRangeThroughputExceeded
- IndexReadKeyRangeThroughputExceeded
- IndexWriteKeyRangeThroughputExceeded

Se ha superado el rendimiento aprovisionado

Estos motivos de limitación se producen cuando las tasas de consumo superan los límites de capacidad aprovisionados en el modo aprovisionado:

- TableReadProvisionedThroughputExceeded
- TableWriteProvisionedThroughputExceeded
- IndexReadProvisionedThroughputExceeded
- IndexWriteProvisionedThroughputExceeded

Se han superado los límites de la cuenta

Estos motivos de limitaciones se producen cuando las tasas de consumo superan las cuotas de rendimiento por cuenta en la región de AWS:

- TableReadAccountLimitExceeded
- TableWriteAccountLimitExceeded
- IndexReadAccountLimitExceeded
- IndexWriteAccountLimitExceeded

Se ha superado el rendimiento máximo bajo demanda

Estos motivos de la limitación se producen cuando las tasas de consumo superan los límites de rendimiento máximo configurados en el modo bajo demanda:

- TableReadMaxOnDemandThroughputExceeded
- TableWriteMaxOnDemandThroughputExceeded
- IndexReadMaxOnDemandThroughputExceeded
- IndexWriteMaxOnDemandThroughputExceeded

1- Se ha superado el rendimiento del rango clave (particiones calientes)

Amazon DynamoDB impone límites de rendimiento específicos por partición, para la tabla y el índice secundario global (GSI). Cada partición tiene un número máximo de unidades de capacidad de lectura (RCU) y unidades de capacidad de escritura (WCU) por segundo. Cuando las particiones reciben un tráfico concentrado que supera estos límites, se limitan, mientras que otras operaciones pueden quedar infrautilizadas, lo que crea “particiones calientes”. La limitación por partición de DynamoDB funciona de forma independiente para las lecturas y escrituras: una partición puede limitar las lecturas mientras las escrituras continúan con normalidad o viceversa. Esta limitación

puede producirse incluso cuando la tabla o el GSI tienen una capacidad total suficiente. Para obtener más información:

- Los límites de partición de DynamoDB y el diseño efectivo de claves de partición abordan la prevención de particiones calientes, consulte Prácticas recomendadas para diseñar claves de partición y utilizarlas con eficacia en DynamoDB.
- Para obtener información sobre conceptos generales de partición y distribución de datos, consulte Particiones en DynamoDB.
- Para obtener más información y escenarios reales para administrar las claves de partición y el rendimiento, consulte the section called “Recursos adicionales”.

Cuando las particiones individuales superan sus límites de rendimiento, DynamoDB devuelve un tipo de motivo de limitación de `KeyRangeThroughputExceeded` en la excepción de limitación. La información identifica que una partición tiene mucho tráfico y qué tipo de operación (lectura o escritura) está causando el problema.

El rendimiento del rango clave superó las medidas de mitigación

Esta sección proporciona una guía para la resolución de situaciones de limitación por partición. Antes de usar esta guía, asegúrese de haber identificado los motivos específicos de limitación a partir del manejo de excepciones de la aplicación y de haber determinado el nombre de recurso de Amazon (ARN) del recurso afectado. Para obtener información sobre cómo recuperar los motivos de la limitación e identificar los recursos limitados, consulte the section called “Marco de diagnóstico”.

Antes de sumergirse en escenarios de limitación específicos, compruebe primero si el problema se resuelve automáticamente:

- DynamoDB se adapta a menudo a las particiones calientes mediante su mecanismo automático de división por calor. Si ve que los eventos de limitación se detienen después de un breve periodo de tiempo, es posible que la tabla ya se haya adaptado al dividir la partición caliente. Cuando las particiones se dividen, cada nueva partición ocupa una sección más pequeña del espacio de claves, lo que puede ayudar a distribuir la carga de manera más uniforme. En muchos casos, no tiene que hacer nada más, ya que DynamoDB ha resuelto el problema automáticamente.

  Para obtener más información sobre el mecanismo de división por calor, consulte the section called “Recursos adicionales”.

Si la limitación persiste, consulte los escenarios de limitación específicos que se indican a continuación para ver las opciones de corrección específicas:

- the section called “TableReadKeyRangeThroughputExceeded”
- the section called “TableWriteKeyRangeThroughputExceeded”
- the section called “IndexReadKeyRangeThroughputExceeded”
- the section called “IndexWriteKeyRangeThroughputExceeded”

TableReadKeyRangeThroughputExceeded

Cuando esto ocurre

La tasa de consumo de una o más particiones de la tabla de DynamoDB supera el límite de rendimiento de lectura de la partición. Esta limitación se produce independientemente de la capacidad aprovisionada total de la tabla y afecta a las tablas aprovisionadas y bajo demanda. Puede supervisar las métricas de CloudWatch en the section called “Diagnóstico y supervisión comunes” para analizar el evento de limitación.

Opciones de corrección

Tenga en cuenta estos pasos para solucionar los problemas de limitación:

Para los modos aprovisionado y bajo demanda:

- Capacidad de precalentamiento: si la limitación persiste, compruebe si la tabla está limitada por su capacidad de the section called “Rendimiento en caliente”. Utilice un rendimiento en caliente o aumente la capacidad de lectura aprovisionada con antelación para los aumentos de tráfico esperados. Al aumentar el rendimiento en caliente, se mejora la capacidad de la tabla para gestionar picos de tráfico repentinos antes de que se produzca una limitación. Con el tiempo, si el rendimiento real se aproxima de forma coherente a los niveles de rendimiento en caliente, DynamoDB puede dividir las particiones ocupadas en función de los patrones de uso observados.
- Identifique las teclas de acceso rápido: si la tabla no lo resolvió automáticamente y el rendimiento en caliente es alto o aumentarlo no ayudó, tendrá que identificar las teclas de acceso rápido específicas. Use the section called “Identificación de teclas de acceso rápido mediante Información·de·colaboradores·de·CloudWatch” para determinar si algún valor de una tecla de partición en particular está caliente. Este es un primer paso para dirigir los esfuerzos de mitigación de manera efectiva. Tenga en cuenta que la identificación no siempre es sencilla, sobre todo en el caso de las particiones calientes móviles (en las que distintas particiones se calientan con el paso

del tiempo) o cuando la limitación se desencadena mediante operaciones como los escaneos. Para estos escenarios complejos, es posible que deba analizar los patrones de acceso de la aplicación y correlacionarlos con el momento en que se producen los eventos de limitación.

- En función del caso de uso, considere la posibilidad de utilizar lecturas coherentes posteriores: cambie de lecturas altamente coherentes a lecturas coherentes posteriores, que consumen la mitad de RCU y pueden duplicar inmediatamente su capacidad de lectura efectiva. Para las prácticas recomendadas sobre cómo implementar lecturas coherentes posteriores para reducir el consumo de capacidad de lectura, consulte the section called “Coherencia de lectura de DynamoDB”.
- Mejore el diseño de las claves de partición: como solución a largo plazo, tenga en cuenta the section called “Mejora del diseño de claves de partición” para distribuir el acceso de manera más uniforme entre las particiones. Este enfoque suele proporcionar la solución más completa a los problemas de partición más candentes, ya que aborda la causa raíz. Sin embargo, requiere una planificación cuidadosa, ya que implica importantes desafíos de migración.

TableWriteKeyRangeThroughputExceeded

Cuando esto ocurre

La tasa de consumo de una o más particiones de la tabla de DynamoDB supera el límite de rendimiento de escritura de la partición. Esta limitación se produce independientemente de la capacidad aprovisionada total de la tabla y afecta a las tablas aprovisionadas y bajo demanda. Puede supervisar las métricas de CloudWatch en the section called “Diagnóstico y supervisión comunes” para analizar el evento de limitación.

Opciones de corrección

Tenga en cuenta estos pasos para solucionar los problemas de limitación:

Para los modos aprovisionado y bajo demanda:

- Capacidad de precalentamiento: si la limitación persiste, compruebe si la tabla está limitada por su capacidad de Descripción del rendimiento en caliente de DynamoDB. Utilice un rendimiento en caliente o aumente la capacidad de escritura aprovisionada con antelación para los aumentos de tráfico esperados. Al aumentar el rendimiento en caliente, se mejora la capacidad de la tabla para gestionar picos de tráfico repentinos antes de que se produzca una limitación. Con el tiempo, si el rendimiento real se aproxima de forma coherente a los niveles de rendimiento en caliente, DynamoDB puede dividir las particiones ocupadas en función de los patrones de uso observados.

- Identifique las teclas de acceso rápido: si la tabla no lo resolvió automáticamente y el rendimiento en caliente es alto o aumentarlo no ayudó, tendrá que identificar las teclas de acceso rápido específicas. Use the section called “Identificación de teclas de acceso rápido mediante Información·de·colaboradores·de·CloudWatch” para determinar si algún valor de una tecla de partición en particular está caliente. Este es un primer paso para dirigir los esfuerzos de mitigación de manera efectiva. Tenga en cuenta estos patrones comunes:
  - Si ve que la misma clave de partición aparece con frecuencia en los datos de limitación, esto indica que se trata de una tecla de acceso rápido concentrada.
  - Si no ve las teclas repetidas, pero escribe los datos de forma ordenada (por ejemplo, marcas temporales secuenciales u operaciones basadas en el escaneo que siguen el orden del espacio de teclas), es probable que tenga particiones dinámicas en las que distintas teclas se calientan con el paso del tiempo a medida que las escrituras se mueven por el espacio de teclas.

  Tenga en cuenta que la limitación de escritura también se puede producir con operaciones como `BatchWriteItem` o transacciones que afectan a varios elementos simultáneamente. Cuando se limitan los elementos individuales de una solicitud de `BatchWriteItem`, DynamoDB no propaga estos errores de limitación al código de la aplicación. En su lugar, DynamoDB devuelve información sobre los elementos no procesados de la respuesta, que la aplicación debe gestionar reintentando esos elementos específicos. Para las transacciones, se produce un error en toda la operación con una `TransactionCanceledException` si algún elemento experimenta limitaciones. Para estos escenarios complejos, es posible que necesite analizar los patrones de escritura y los flujos de trabajo de ingesta de datos de la aplicación, correlacionarlos con el momento en que se producen los eventos de limitación e implementar las estrategias adecuadas de gestión de los reintentos.
- Mejore el diseño de las claves de partición: como solución a largo plazo, tenga en cuenta the section called “Mejora del diseño de claves de partición” para distribuir el acceso de manera más uniforme entre las particiones. Este enfoque suele proporcionar la solución más completa a los problemas de partición más candentes, ya que aborda la causa raíz. Sin embargo, requiere una planificación cuidadosa, ya que implica importantes desafíos de migración.

IndexReadKeyRangeThroughputExceeded

Cuando esto ocurre

La tasa de consumo de una o más particiones del GSI de DynamoDB supera el límite de rendimiento de lectura de la partición. Esta limitación se produce independientemente de la capacidad aprovisionada total del GSI y afecta a las tablas aprovisionadas y bajo demanda. Puede supervisar

las métricas de CloudWatch en the section called “Diagnóstico y supervisión comunes” para analizar el evento de limitación.

Opciones de corrección

Tenga en cuenta estos pasos para solucionar los problemas de limitación:

- Capacidad de precalentamiento: si la limitación persiste, compruebe si el GSI está limitado por su capacidad de Descripción del rendimiento en caliente de DynamoDB. Utilice un rendimiento en caliente o aumente la capacidad de lectura aprovisionada con antelación para los aumentos de tráfico esperados. Al aumentar el rendimiento en caliente, se mejora la capacidad del GSI para gestionar picos de tráfico repentinos antes de que se produzca una limitación. Con el tiempo, si el rendimiento real se aproxima de forma coherente a los niveles de rendimiento en caliente, DynamoDB puede dividir las particiones ocupadas en función de los patrones de uso observados.
- Identifique las teclas de acceso rápido: si el GSI no lo resolvió automáticamente y el rendimiento en caliente es alto o aumentarlo no ayudó, tendrá que identificar las teclas de acceso rápido específicas. Use the section called “Identificación de teclas de acceso rápido mediante Información·de·colaboradores·de·CloudWatch” para determinar si algún valor de una tecla de partición en particular está caliente. Este es un primer paso para dirigir los esfuerzos de mitigación de manera efectiva. Tenga en cuenta que, en el caso de los GSI, la distribución de claves de partición puede diferir considerablemente de la tabla base, lo que crea diferentes patrones de teclas de acceso rápido.
- Rediseñe las claves de partición de GSI: considere si el diseño de la clave de GSI podría estar creando puntos calientes artificiales (como indicadores de estado, claves de solo fecha o atributos booleanos) que concentren las lecturas en un número reducido de particiones. Considere la posibilidad de utilizar claves compuestas que combinen el atributo de baja cardinalidad con un atributo de alta cardinalidad (por ejemplo, “ACTIVE#customer123” en lugar de simplemente “ACTIVE”) o aplique técnicas de the section called “Partición de escritura” a los elementos de la tabla base que afecten a la distribución de GSI para distribuir las escrituras en varias particiones. Aunque la consulta de datos particionados requiere una lógica de aplicación adicional para agregar los resultados, este enfoque evita la limitación al distribuir los patrones de acceso de manera más uniforme.

IndexWriteKeyRangeThroughputExceeded

Cuando esto ocurre

La tasa de consumo de una o más particiones del GSI de DynamoDB supera el límite de rendimiento de escritura de la partición. Esta limitación se produce independientemente de la capacidad aprovisionada total del GSI y afecta a las tablas aprovisionadas y bajo demanda. Puede supervisar las métricas de CloudWatch en the section called “Diagnóstico y supervisión comunes” para analizar el evento de limitación.

Opciones de corrección

Tenga en cuenta estos pasos para solucionar los problemas de limitación:

- Rediseñe la clave de partición de GSI: revise el diseño de la clave de partición de GSI para comprobar que tiene la cardinalidad (unicidad) suficiente para distribuir las escrituras de manera uniforme. Una causa común de la limitación de la escritura de GSI es el uso de atributos de baja cardinalidad como claves de partición de GSI (por ejemplo, indicadores de estado con solo unos pocos valores posibles). Aunque la tabla base tenga claves de partición bien distribuidas, el GSI puede seguir teniendo particiones calientes si la clave de partición concentra las escrituras en un número reducido de valores. Por ejemplo, si el 80 % de los elementos tienen “status=ACTIVE”, se crea una partición muy caliente en un GSI basado en el estado. Considere la posibilidad de utilizar claves compuestas que combinen el atributo de baja cardinalidad con un atributo de alta cardinalidad (por ejemplo, “ACTIVE#customer123” en lugar de simplemente “ACTIVE”) o aplique técnicas de the section called “Partición de escritura” a los elementos de la tabla base que afecten a la distribución de GSI para distribuir las escrituras en varias particiones. Aunque la consulta de datos particionados requiere una lógica de aplicación adicional para agregar los resultados, este enfoque evita la limitación al distribuir los patrones de acceso de manera más uniforme.
- Capacidad de precalentamiento: compruebe si el GSI está limitado por su capacidad de Descripción del rendimiento en caliente de DynamoDB. Utilice un rendimiento en caliente o aumente la capacidad de lectura aprovisionada con antelación para los aumentos de tráfico esperados. Al aumentar el rendimiento en caliente, se mejora la capacidad del GSI para gestionar picos de tráfico repentinos antes de que se produzca una limitación. Con el tiempo, si el rendimiento real se aproxima de forma coherente a los niveles de rendimiento en caliente, DynamoDB puede dividir las particiones ocupadas en función de los patrones de uso observados.
- Optimice las proyecciones de GSI: aplique técnicas de the section called “Optimización de las proyecciones del GSI” para reducir el volumen de escritura en los GSI. Proyectar menos atributos puede reducir considerablemente la capacidad de escritura consumida por cada actualización del GSI.

Diagnóstico y supervisión comunes

Al solucionar problemas de limitación por partición, varias métricas de CloudWatch pueden ayudar a identificar las particiones calientes y confirmar la causa raíz.

Métricas esenciales de CloudWatch

Supervise estas métricas clave para diagnosticar la limitación por partición:

- Eventos de limitación por partición: `ReadKeyRangeThroughputThrottleEvents` y `WriteKeyRangeThroughputThrottleEvents` realizan un seguimiento de los casos en los que las particiones individuales superan sus límites de rendimiento. `ReadThrottleEvents` y `WriteThrottleEvents` realizan un seguimiento de los casos en que las solicitudes de lectura o escritura superen la capacidad aprovisionada.
- Consumo de capacidad: `ConsumedReadCapacityUnits` y `ConsumedWriteCapacityUnits` muestran los patrones de uso generales.

Procedimientos de resolución

Identificación de teclas de acceso rápido mediante Información·de·colaboradores·de·CloudWatch

Utilice este procedimiento para identificar qué claves de partición están provocando la limitación.

1. Habilite Información·de·colaboradores·de·CloudWatch en la tabla o GSI para realizar un seguimiento de las claves más limitadas. Considere la posibilidad de mantener Información·de·colaboradores·de·CloudWatch habilitado de forma continua para recibir alertas de limitación en tiempo real mediante el modo Claves limitadas. Este modo se centra exclusivamente en las solicitudes limitadas y solo procesa los eventos cuando se produce una limitación. Esta supervisión específica es una forma rentable de mantener una visibilidad continua de los problemas de limitación.
2. Identifique qué claves están causando los problemas con las particiones calientes.
3. (Si está habilitado el modo completo de claves a las que se ha accedido y limitadas) Analice los patrones de acceso a lo largo del tiempo para determinar si las teclas de acceso rápido son coherentes o se utilizan durante periodos específicos.

Mejora del diseño de claves de partición

Utilice este enfoque cuando pueda modificar el esquema de la tabla para distribuir mejor el tráfico entre las particiones. Siempre que sea posible, esta es la solución a largo plazo más eficaz para los

problemas de particiones calientes. Lo ideal sería considerar cuidadosamente el diseño de la clave de partición durante la fase inicial de diseño de la tabla.

El rediseño de la clave de partición representa un cambio fundamental en el modelo de datos que afecta a todo el ecosistema de aplicaciones. Antes de continuar con este enfoque, considere detenidamente estas importantes limitaciones:

- Complejidad de la migración de datos: el rediseño de las claves de partición requiere la migración de todos los datos existentes, lo que puede requerir muchos recursos y tiempo en el caso de tablas grandes.
- Cambios en el código de la aplicación: todo el código de la aplicación que lee o escribe en la tabla se debe actualizar para utilizar la nueva estructura de claves.
- Impacto en la producción: la migración a un nuevo diseño de clave suele requerir tiempo de inactividad o estrategias complejas de escritura doble durante la transición.

Para obtener una guía y principios completos sobre el diseño de claves de partición, consulte the section called “Diseño de claves de partición” y the section called “Distribución de cargas de trabajo”.

Optimización de las proyecciones del GSI

Revise los patrones de consulta de la aplicación para determinar exactamente qué atributos deben estar disponibles al consultar el GSI y limite las proyecciones solo a esos atributos. Al actualizar los atributos que no están proyectados en un GSI, no se produce ninguna operación de escritura en ese GSI, lo que reduce el consumo de rendimiento de escritura durante las actualizaciones. Esta estrategia de proyección específica optimiza el rendimiento y el costo y, al mismo tiempo, cumple con los requisitos de consulta de la aplicación. Tenga en cuenta que proyectar menos atributos reduce el consumo de capacidad de escritura, pero puede requerir lecturas de tablas base adicionales.

Para obtener más información sobre las estrategias de proyección eficientes, consulte Prácticas recomendadas para utilizar índices secundarios en DynamoDB.

Recursos adicionales

En las siguientes publicaciones del blog se proporcionan ejemplos prácticos y detalles prácticos de los conceptos que se tratan en esta guía:

- Para obtener más información sobre el uso de los GSI para distribuir el tráfico de lectura, consulte Using Global Secondary Indexes to create eventually consistent secondary indexes in Amazon DynamoDB.

- Para obtener orientación práctica sobre cómo escalar DynamoDB y administrar particiones calientes, consulte Parte 1: Escalado de DynamoDB: cómo afectan al rendimiento las particiones, las teclas de acceso rápido y la división por calor.
- Para obtener información detallada sobre el funcionamiento del mecanismo de división por calor de DynamoDB, sus beneficios y los detalles de la implementación, consulte Part 3: Summary and best practices.
- Para obtener estrategias detalladas de partición de escritura, consulte the section called “Partición de escritura”.

2- Se ha superado el rendimiento aprovisionado

La limitación de la capacidad aprovisionada se produce cuando la tasa de consumo de la aplicación supera las unidades de capacidad de lectura o escritura (RCU/WCU) configuradas para las tablas o los índices secundarios globales. Aunque DynamoDB proporciona una capacidad de ampliación para gestionar picos de tráfico ocasionales, las solicitudes continuas que superan los límites aprovisionados provocan una limitación. Cuando esto ocurre, DynamoDB devuelve un tipo de motivo de limitación `ProvisionedThroughputExceeded` en la excepción de limitación. El motivo identifica si el problema está relacionado con las operaciones de lectura o escritura y si afecta a la tabla base o a un índice secundario global.

La limitación se puede producir independientemente de si el escalado automático está activado. El escalado automático se adapta a los aumentos de consumo, pero no responde al instante y está limitado por los límites de capacidad máxima que configure. Esto significa que aún se pueden producir limitaciones durante picos de tráfico repentinos o cuando el consumo supera los límites máximos de escalado automático.

El rendimiento aprovisionado ha superado las medidas de mitigación

Esta sección proporciona una guía para la resolución de situaciones de limitación de capacidad aprovisionada. Antes de usar esta guía, asegúrese de haber identificado el motivo específico de la limitación a partir del manejo de excepciones de la aplicación y de haber determinado el nombre de recurso de Amazon (ARN) del recurso afectado. Para obtener información sobre cómo recuperar los motivos de la limitación e identificar los recursos limitados, consulte the section called “Marco de diagnóstico”.

Antes de sumergirse en escenarios de limitación específicos, primero considere si la limitación es realmente un problema que necesita solución:

- En aplicaciones de DynamoDB bien optimizadas, es normal y se espera que se produzcan limitaciones ocasionales. La limitación simplemente significa que está consumiendo el 100 % de lo que ha aprovisionado. Si la aplicación gestiona la limitación sin problemas con los reintentos y el rendimiento general cumple con los requisitos, es posible que la limitación no requiera una acción inmediata.
- Sin embargo, si la limitación provoca una latencia inaceptable del cliente, degrada la experiencia del usuario o impide que las operaciones críticas se completen a tiempo, proceda con las siguientes opciones de mitigación.

Cuando necesite solucionar problemas de limitación, primero determine si la limitación se debe a:

- Picos de tráfico temporales: incrementos de tráfico de corta duración que superan la capacidad aprovisionada pero que no se mantienen. Estos requieren estrategias diferentes a las del tráfico alto continuo.
- Tráfico alto continuo: cargas de trabajo sostenidas que superan de forma coherente la capacidad aprovisionada.

Para los picos de tráfico, considere las estrategias del blog Handle traffic spikes with Amazon DynamoDB provisioned capacity en the section called “Recursos adicionales”.

Para un tráfico elevado y continuo, considere las siguientes opciones de ajuste de capacidad:

- the section called “TableReadProvisionedThroughputExceeded”
- the section called “TableWriteProvisionedThroughputExceeded”
- the section called “IndexReadProvisionedThroughputExceeded”
- the section called “IndexWriteProvisionedThroughputExceeded”

TableReadProvisionedThroughputExceeded

Cuando esto ocurre

La tasa de consumo de lectura de la aplicación supera las unidades de capacidad de lectura (RCU) aprovisionadas configuradas para la tabla. Puede supervisar las métricas de CloudWatch en the section called “Diagnóstico y supervisión comunes” para analizar el evento de limitación.

Enfoque de resolución

Tenga en cuenta estas estrategias para resolver la limitación de la capacidad de lectura:

- Cambie al modo de capacidad bajo demanda: considere la posibilidad de cambiar la tabla a modo bajo demanda si experimenta una limitación frecuente debido a los picos de tráfico. Bajo demanda elimina las preocupaciones de aprovisionamiento y se escala automáticamente con la carga de trabajo.
- Si permanece en el modo aprovisionado y escalado automático no está habilitado:
  - Considere aumentar la capacidad de lectura de la tabla.
  - Habilite el escalado automático para la capacidad de lectura en la tabla.
- Si el escalado automático está habilitado (predeterminado para las tablas creadas en la consola):
  - Optimice los parámetros de lectura de escalado automático de la tabla.

TableWriteProvisionedThroughputExceeded

Cuando esto ocurre

La tasa de consumo de escritura de la aplicación supera las unidades de capacidad de escritura (WCU) aprovisionadas configuradas para la tabla. Puede supervisar las métricas de CloudWatch en the section called “Diagnóstico y supervisión comunes” para analizar el evento de limitación.

Enfoque de resolución

Tenga en cuenta estas estrategias para resolver la limitación de la capacidad de escritura:

- Cambie al modo de capacidad bajo demanda: considere la posibilidad de cambiar la tabla a modo bajo demanda si experimenta una limitación frecuente debido a los picos de tráfico. Bajo demanda elimina las preocupaciones de aprovisionamiento y se escala automáticamente con la carga de trabajo.
- Si permanece en el modo aprovisionado y escalado automático no está habilitado:
  - Considere aumentar la capacidad de escritura de la tabla.
  - Habilite el escalado automático para la capacidad de escritura en la tabla.
- Si el escalado automático está habilitado (predeterminado para las tablas creadas en la consola):
  - Optimice los parámetros de escritura de escalado automático de la tabla.

IndexReadProvisionedThroughputExceeded

Cuando esto ocurre

El consumo de lectura en un índice secundario global (GSI) supera las unidades de capacidad de lectura (RCU) aprovisionadas del GSI. Puede supervisar las métricas de CloudWatch en the section called “Diagnóstico y supervisión comunes” para analizar el evento de limitación.

Enfoque de resolución

Tenga en cuenta estas estrategias para resolver la limitación de la capacidad de lectura de GSI:

- Cambie al modo de capacidad bajo demanda: considere cambiar la tabla base a bajo demanda si experimenta limitaciones frecuentes debido a los picos de tráfico. Bajo demanda elimina las preocupaciones de aprovisionamiento y se escala automáticamente con la carga de trabajo.
- Si permanece en el modo aprovisionado y escalado automático no está habilitado:
  - Considere aumentar la capacidad de lectura del GSI.
  - Habilite el escalado automático para la capacidad de lectura en el GSI.
- Si el escalado automático está habilitado (predeterminado para las tablas creadas en la consola):
  - Optimice los parámetros de lectura de escalado automático de GSI.

IndexWriteProvisionedThroughputExceeded

Cuando esto ocurre

Las actualizaciones de los elementos de la tabla base desencadenan las escrituras en un GSI que superan la capacidad de escritura aprovisionada de GSI. Esto provoca una limitación de contrapresión en las escrituras de la tabla base. Puede supervisar las métricas de CloudWatch en the section called “Diagnóstico y supervisión comunes” para analizar el evento de limitación.

Enfoque de resolución

Tenga en cuenta estas estrategias para resolver la limitación de la capacidad de escritura de GSI:

- Cambie al modo de capacidad bajo demanda: considere cambiar la tabla base a bajo demanda si experimenta limitaciones frecuentes debido a los picos de tráfico. Bajo demanda elimina las preocupaciones de aprovisionamiento y se escala automáticamente con la carga de trabajo.
- Si permanece en el modo aprovisionado y escalado automático no está habilitado:
  - Considere aumentar la capacidad de escritura de GSI.
  - Habilite el escalado automático para la capacidad de escritura en el GSI.
- Si el escalado automático está habilitado (predeterminado para las tablas creadas en la consola):
  - Optimice los parámetros de escritura de escalado automático de GSI.

Diagnóstico y supervisión comunes

Al solucionar problemas de rendimiento, varias métricas de CloudWatch pueden ayudar a identificar la causa raíz.

Métricas esenciales de CloudWatch

Supervise estas métricas de claves para diagnosticar la limitación de la capacidad aprovisionada:

- Eventos de limitación: `ReadProvisionedThroughputThrottleEvents` y `WriteProvisionedThroughputThrottleEvents` realizan un seguimiento cuando las solicitudes se limitan debido a este motivo. `ReadThrottleEvents` y `WriteThrottleEvents` realizan un seguimiento cuando las solicitudes de lectura o escritura superan la capacidad aprovisionada.
- Consumo de capacidad: `ConsumedReadCapacityUnits` y `ConsumedWriteCapacityUnits` muestran el uso real.
- Capacidad aprovisionada: `ProvisionedReadCapacityUnits` y `ProvisionedWriteCapacityUnits` muestran los límites configurados.

Procedimientos de resolución

Aumento de la capacidad de rendimiento de la tabla

Utilice este procedimiento cuando el escalado automático no esté habilitado y necesite un aumento de capacidad inmediato.

1. Actualice la capacidad aprovisionada de la tabla mediante la consola de DynamoDB, la AWS CLI o el SDK:
   - Para la capacidad de lectura: aumente el parámetro `ReadCapacityUnits`, que especifica el número máximo de lecturas altamente coherentes que se consumen por segundo antes de que DynamoDB limite las solicitudes.
   - Para la capacidad de escritura: aumente el parámetro `WriteCapacityUnits`, que especifica el número máximo de escrituras que se consumen por segundo antes de que DynamoDB limite las solicitudes.
2. Compruebe que la nueva configuración de capacidad no supere las cuotas de rendimiento por tabla y que el consumo total de la cuenta se mantenga por debajo de las cuotas de rendimiento por cuenta de la región. Si se está acercando a estos límites, considere cambiar al modo de capacidad bajo demanda en su lugar.

Configuración del escalado automático de la tabla para ajustar la capacidad de lectura o escritura de la tabla o GSI

Configure el escalado automático de DynamoDB para ajustar automáticamente la capacidad de lectura o escritura en función de los patrones de tráfico. Puede configurar el escalado automático de forma independiente para las tablas y para los GSI, con controles independientes para las unidades de capacidad de lectura y escritura.

1. Habilite el escalado automático para la capacidad de lectura, capacidad de escritura o ambas en la tabla o GSI.
2. Establezca un porcentaje de uso objetivo con margen de maniobra para los picos de tráfico.

> **Note**
>
> Un uso objetivo menor aumenta los costos y la frecuencia de escalado. Los objetivos por debajo del 40 % pueden provocar un sobreaprovisionamiento. Supervise los patrones de uso y los costos para equilibrar el rendimiento y la eficiencia.

3. Establecimiento de los límites de capacidad:
   - Mínimo de RCU/WCU: mantiene una capacidad suficiente durante los periodos de tráfico bajo.
   - Máximo de RCU/WCU: se adapta a las demandas de picos de tráfico y protege contra los eventos de escalado descontrolado.

Para obtener instrucciones sobre la configuración y la administración del escalado automático de DynamoDB, consulte Administración automática de la capacidad de rendimiento con el escalado automático de DynamoDB.

> **Note**
>
> Por lo general, el escalado automático tarda varios minutos en responder a los cambios de tráfico. En caso de picos de tráfico repentinos, la capacidad de ampliación de la tabla proporciona protección inmediata mientras el escalado automático se ajusta. Configure el uso objetivo con el margen de maniobra adecuado para disponer de tiempo para escalar las operaciones y preservar la capacidad de ampliación ante una demanda imprevista.

Optimización de los ajustes de escalado automático de lectura o escritura de la tabla o índice

Utilice este procedimiento cuando el escalado automático esté habilitado pero se siga produciendo la limitación. Puede ajustar el escalado automático de forma independiente para las tablas y para los índices secundarios globales (GSI), con controles independientes para las unidades de capacidad de lectura y escritura.

- Ajuste el uso objetivo: considere reducir el uso objetivo de la tabla o los GSI para desencadenar el escalado antes de que se produzca la limitación. Asegúrese de supervisar el tráfico después de realizar estos ajustes. Consulte the section called “Configuración del escalado automático de la tabla para ajustar la capacidad de lectura o escritura de la tabla o GSI” para obtener más información sobre el consumo de capacidad y las implicaciones de costos.
- Revise los límites de capacidad: asegúrese de que los ajustes de capacidad mínima y máxima se ajusten a los patrones de carga de trabajo reales.

Cambio al modo de capacidad bajo demanda

Para obtener información general sobre el cambio de los modos de capacidad, consulte the section called “Cambio de los modos de capacidad”. Consulte Service Quotas para obtener información sobre las restricciones específicas al cambiar de modo.

Aumento de la capacidad de rendimiento de GSI

Utilice este procedimiento cuando el escalado automático no esté habilitado en el GSI o necesite un aumento de capacidad inmediato.

1. Actualice la capacidad aprovisionada de GSI mediante la consola de DynamoDB, la AWS CLI o el SDK:
   - Para la capacidad de lectura: aumente el parámetro `ReadCapacityUnits` para el GSI específico, que especifica el número máximo de lecturas que el GSI puede consumir por segundo antes de que DynamoDB limite las solicitudes. Tenga en cuenta que los GSI solo admiten lecturas coherentes posteriores.
   - Para la capacidad de escritura: aumente el parámetro `WriteCapacityUnits` para el GSI específico, que especifica el número máximo de escrituras que el GSI puede consumir por segundo antes de que DynamoDB limite las solicitudes.
2. Asegúrese de que la capacidad de rendimiento aprovisionada de GSI se mantenga dentro de las cuotas de rendimiento por cuenta y por tabla.

Recursos adicionales

- Para obtener información detallada sobre cómo gestionar los picos de tráfico en las tablas de capacidad aprovisionada de DynamoDB, incluidas varias estrategias, desde el uso del escalado automático y la capacidad de ampliación hasta la administración estratégica de la limitación, consulte Handle traffic spikes with Amazon DynamoDB provisioned capacity.
- Para obtener información sobre cómo utilizar una expresión cron para programar una política de escalado, consulte Optimize costs by scheduling provisioned capacity for DynamoDB.
- Para obtener información práctica sobre la supervisión y el análisis de los patrones de uso del rendimiento de las tablas de DynamoDB en el modo de capacidad aprovisionada, consulte Cómo evaluar la utilización del rendimiento de las tablas de Amazon DynamoDB en el modo aprovisionado.

3- Se han superado los límites de la cuenta

Las tablas bajo demanda no tienen niveles de capacidad aprovisionados que administrar, pero DynamoDB impone límites de rendimiento por cuenta para evitar una ejecución descontrolada y garantizar un uso justo de los recursos por parte de todos los clientes. Estos límites de cuenta por tabla sirven como medidas de seguridad ajustables y se establecen para cada combinación de cuentas y regiones. Cuando la tasa de consumo de lectura o escritura supera estos límites, DynamoDB devuelve un tipo de motivo de limitación `AccountLimitExceeded` en la excepción de limitación. Los límites de cuenta predeterminados por tabla se aplican automáticamente cuando las tablas no tienen configurados ajustes de rendimiento máximo personalizados. Si lo desea, puede configurar los ajustes de rendimiento máximo para controlar los costos y mejorar la previsibilidad, o solicitar aumentos de cuota a través de la consola de the section called “Cuotas” si los requisitos de la aplicación superan los límites predeterminados.

El límite de la cuenta superó las medidas de mitigación

Esta sección proporciona una guía para la resolución de situaciones de limitación del límite de las cuentas. Antes de usar esta guía, asegúrese de haber identificado los motivos específicos de limitación a partir del manejo de excepciones de la aplicación y de haber determinado el nombre de recurso de Amazon (ARN) del recurso afectado. Para obtener información sobre cómo recuperar los motivos de la limitación e identificar los recursos limitados, consulte the section called “Marco de diagnóstico”.

Antes de sumergirse en escenarios de limitación específicos, primero determine si realmente es necesario tomar medidas:

- Evalúe el impacto en el rendimiento: compruebe si la aplicación sigue cumpliendo sus requisitos de rendimiento a pesar de la limitación. Muchas aplicaciones funcionan correctamente dentro o cerca de los límites de la cuenta, especialmente durante las operaciones masivas o las migraciones de datos.
- Revise los patrones de limitación: si la limitación es intermitente y la aplicación gestiona los reintentos de forma eficaz, los límites actuales pueden ser suficientes para la carga de trabajo.

Si el rendimiento de la aplicación es aceptable, incluso aunque en ocasiones supere los límites de la cuenta, puede elegir simplemente supervisar la situación en lugar de implementar cambios inmediatos.

Si determina que la limitación está provocando problemas de rendimiento inaceptables o problemas de fiabilidad, seleccione un motivo de limitación específico a continuación para ver las opciones de mitigación recomendadas:

- the section called “TableReadAccountLimitExceeded”
- the section called “TableWriteAccountLimitExceeded”
- the section called “IndexReadAccountLimitExceeded”
- the section called “IndexWriteAccountLimitExceeded”

TableReadAccountLimitExceeded

Cuando esto ocurre

El consumo de lectura de la tabla ha superado la cuota de rendimiento de lectura por tabla por cuenta para la región. Puede supervisar las métricas de CloudWatch en the section called “Diagnóstico y supervisión comunes” para analizar el evento de limitación.

Enfoque de resolución

Siga los pasos siguientes para solucionar esta limitación:

- Solicite aumentos de cuota:

  Solicite un aumento del límite de rendimiento de lectura por tabla (código de cuota L-CF0CBE56). Para ver pasos detallados acerca de cómo enviar la solicitud, consulte Solicitud de aumentos de cuota por tabla.

TableWriteAccountLimitExceeded

Cuando esto ocurre

El consumo de escritura de la tabla ha superado la cuota de rendimiento de escritura por tabla por cuenta para la región. Puede supervisar las métricas de CloudWatch en the section called “Diagnóstico y supervisión comunes” para analizar el evento de limitación.

Enfoque de resolución

Siga los pasos siguientes para solucionar esta limitación:

- Solicite aumentos de cuota: solicite un aumento del límite de rendimiento de escritura por tabla (código de cuota L-AB614373). Para ver pasos detallados acerca de cómo enviar la solicitud, consulte Solicitud de aumentos de cuota por tabla.

IndexReadAccountLimitExceeded

Cuando esto ocurre

Las operaciones de lectura dirigidas a un índice secundario global (GSI) consumen más rendimiento del que permite la cuota de lectura por tabla de la cuenta en la región de AWS actual. La cuota de rendimiento de lectura por tabla por cuenta se aplica de forma conjunta a una tabla y a todos sus GSI juntos. Puede supervisar las métricas de CloudWatch en the section called “Diagnóstico y supervisión comunes” para analizar el evento de limitación.

Enfoque de resolución

Elija la resolución adecuada en función de la distribución de la capacidad de la cuenta:

- Solicite aumentos de cuota: solicite un aumento del límite de rendimiento de lectura por tabla (código de cuota L-CF0CBE56). Para ver pasos detallados acerca de cómo enviar la solicitud, consulte Solicitud de aumentos de cuota por tabla.
- Optimice el uso del GSI: revise los patrones de diseño y consulta del GSI para reducir el consumo innecesario de capacidad de lectura.

IndexWriteAccountLimitExceeded

Cuando esto ocurre

Las operaciones de escritura en la tabla base generan las actualizaciones correspondientes en un GSI que, en conjunto, superan la cuota de rendimiento de escritura por tabla por cuenta de la región de AWS. Cada escritura en un elemento de la tabla base que contiene atributos indexados por un GSI desencadena una operación de escritura correspondiente a ese GSI. Estas operaciones de escritura combinadas se tienen en cuenta para la cuota de rendimiento de escritura por tabla. Puede supervisar las métricas de CloudWatch en the section called “Diagnóstico y supervisión comunes” para analizar los patrones y el tiempo de estos eventos de limitación e identificar qué operaciones están causando la excesiva actividad de escritura de GSI.

Enfoque de resolución

Elija la resolución adecuada en función de la distribución de la capacidad de la cuenta:

- Solicite aumentos de cuota: solicite un aumento del límite de rendimiento de escritura por tabla (código de cuota L-AB614373) para dar cabida a un mayor tráfico de escritura de GSI procedente de las operaciones de la tabla base. La cuota de rendimiento de escritura por tabla se aplica a toda la tabla, incluidos todos los GSI. Para ver pasos detallados acerca de cómo enviar la solicitud, consulte Solicitud de aumentos de cuota por tabla.
- Optimice las proyecciones del GSI: revise las proyecciones y el diseño del GSI para reducir el volumen de escritura de los GSI.

Diagnóstico y supervisión comunes

Al solucionar problemas por superación del límite de la cuenta, varias métricas de CloudWatch pueden ayudar a identificar si se están alcanzando los límites por tabla o por cuenta y a comprender los patrones de distribución de la capacidad.

Métricas esenciales de CloudWatch

Supervise estas métricas clave para diagnosticar la limitación de los límites de las cuentas:

- Eventos de limitación de límites de cuentas: `ReadAccountLimitThrottleEvents` y `WriteAccountLimitThrottleEvents` realizan un seguimiento cuando las solicitudes se limitan debido a los límites por cuenta. `ReadThrottleEvents` y `WriteThrottleEvents` realizan un seguimiento de los casos en que las solicitudes de lectura o escritura superan la capacidad aprovisionada.
- Consumo de capacidad por recurso: `ConsumedReadCapacityUnits` y `ConsumedWriteCapacityUnits` para cada tabla y GSI muestran el uso individual de los recursos.

- Consumo en toda la cuenta: use expresiones matemáticas de métricas de CloudWatch para sumar la capacidad consumida en todas las tablas y los GSI para supervisar el uso total de la cuenta.

Procedimientos de resolución

Solicitud de aumentos de cuota por tabla

Si las aplicaciones deben funcionar más allá de los límites de rendimiento actuales por tabla, debe enviar una solicitud de aumento de cuota mediante el procedimiento que se indica a continuación. Cada tabla de DynamoDB en la cuenta de AWS (junto con todos sus GSI asociados) está sujeta a estas cuotas de rendimiento dentro de una región específica. Estas cuotas representan la capacidad máxima de lectura o escritura que cualquier tabla individual y sus GSI pueden consumir de forma conjunta, y se aplican de forma independiente a cada tabla y no de forma agregada a todas las tablas de la cuenta.

Si lo desea, también puede establecer límites inferiores por tabla o por GSI configurando sus ajustes de rendimiento máximo bajo demanda.

1. Identifique la cuota específica que necesita aumentar:
   - Límite de rendimiento de lectura por tabla (código de cuota L-CF0CBE56): 40 000 RCU por tabla de forma predeterminada
   - Límite de rendimiento de escritura por tabla (código de cuota L-AB614373): 40 000 WCU por tabla de forma predeterminada
2. Use la consola de AWS Service Quotas para solicitar un aumento:
   - Navegación hasta el servicio de DynamoDB en Service Quotas
   - Búsqueda de la cuota adecuada mediante el código de cuota
   - Solicitud de un aumento en función del consumo máximo proyectado
3. Proporcione una justificación del aumento, que incluya:
   - Patrones de uso actuales y requisitos de tráfico máximo
   - Justificación empresarial para la capacidad aumentada
   - Plazo para determinar cuándo se necesita aumentar la capacidad

> **Note**
>
> Los aumentos de cuotas normalmente tardan entre 24 y 48 horas en procesarse. Planifique las solicitudes en consecuencia y tenga en cuenta estrategias de mitigación temporales mientras espera su aprobación.

Optimización de las proyecciones y el diseño del GSI

Optimice las proyecciones y el diseño del índice secundario global (GSI) para reducir el consumo de capacidad y mejorar el rendimiento.

Estrategias de proyección selectivas

Si las consultas solo necesitan acceder a unos pocos atributos, proyectar solo esos atributos reduce la cantidad de datos que se escriben en el GSI cuando cambian los elementos de la tabla base. Para obtener información sobre los tipos de proyecciones, consulte Proyecciones para índices secundarios globales.

1. Analice los patrones de consulta: revise los patrones de consulta de la aplicación para identificar a qué atributos se accede realmente a través del GSI.
2. Utilice proyecciones selectivas: proyecte solo los atributos que realmente se necesitan en las consultas para reducir el volumen de escritura.
3. Tenga en cuenta `KEYS_ONLY`: si las consultas solo necesitan los atributos clave, utilice la proyección de `KEYS_ONLY` para minimizar el volumen de escritura.
4. Encuentre el equilibrio entre lectura y escritura: proyectar menos atributos reduce el consumo de capacidad de escritura, pero puede requerir lecturas de tablas base adicionales.

Implementación de GSI dispersos

Los GSI dispersos contienen solo elementos que tienen el atributo indexado, en lugar de todos los elementos de la tabla base. Esto reduce la densidad de particiones y mejora el rendimiento cuando se consultan con frecuencia subconjuntos de datos específicos.

1. Diseñe GSI que solo incluyan elementos con valores de atributo específicos.
2. Implemente la indexación condicional configurando solo el atributo de clave de partición de GSI en los elementos que se deben indexar.

3. Utilice claves compuestas en GSI dispersos (por ejemplo, status#timestamp) para distribuir aún más el tráfico dentro del subconjunto de elementos indexados.

Para obtener más información sobre la implementación de estas estrategias, consulte Prácticas recomendadas para utilizar índices secundarios en DynamoDB.

4- Se ha superado el rendimiento máximo bajo demanda

Al configurar una tabla bajo demanda o un GSI, puede establecer opcionalmente un límite de rendimiento máximo (`MaxReadRequestUnits` y `MaxWriteRequestUnits`) por tabla o índice para evitar costos desmesurados o proteger los sistemas posteriores de verse abrumados. Para obtener más información acerca del rendimiento máximo, consulte the section called “Rendimiento máximo de DynamoDB para las tablas bajo demanda”.

Cuando el consumo de lectura o escritura supera estos límites autoimpuestos, las solicitudes adicionales que superan el límite reciben respuestas rápidas de limitación. DynamoDB devuelve excepciones con un tipo de motivo de limitación `MaxOnDemandThroughputExceeded`, que indica qué recurso ha alcanzado su límite de rendimiento.

Limitación de rendimiento máximo bajo demanda superado

En esta sección, se proporciona una guía de resolución para situaciones de superación del límite de rendimiento máximo bajo demanda. Antes de usar esta guía, asegúrese de haber identificado los motivos específicos de limitación a partir del manejo de excepciones de la aplicación y de haber determinado el nombre de recurso de Amazon (ARN) del recurso afectado. Para obtener información sobre cómo recuperar los motivos de la limitación e identificar los recursos limitados, consulte the section called “Marco de diagnóstico”.

Antes de sumergirse en escenarios de limitación específicos, primero considere si realmente es necesario tomar medidas:

- Evalúe la configuración de rendimiento máximo: estos límites se configuraron intencionalmente para controlar los costos o proteger los sistemas posteriores. Si recibe eventos de limitación de `MaxOnDemandThroughputExceeded`, los límites funcionan según lo diseñado. Considere si aumentar estos límites se ajusta a sus objetivos originales de control de costos o de protección del sistema.
- Evalúe el impacto de la aplicación: determine si la limitación realmente está causando problemas a las aplicaciones o a los usuarios. Si las aplicaciones gestionan los reintentos de forma eficaz y

cumplen los requisitos de rendimiento a pesar de las limitaciones ocasionales, mantener los límites actuales podría ser la opción adecuada.

- Revise los patrones de tráfico: analice si la limitación representa un patrón de tráfico esperado o un pico inusual. Si detecta patrones de tráfico predecibles y recurrentes que superan los límites de forma coherente, podría estar justificado ajustar la configuración de rendimiento máximo. En el caso de picos temporales, implementar mejores estrategias de distribución de solicitudes podría ser más apropiado que aumentar los límites.

Si, tras considerarlo, determina que es necesario ajustar la configuración de rendimiento máximo, consulte los siguientes escenarios de limitación específicos para ver las opciones de corrección específicas:

- the section called “TableReadMaxOnDemandThroughputExceeded”
- the section called “TableWriteMaxOnDemandThroughputExceeded”
- the section called “IndexReadMaxOnDemandThroughputExceeded”
- the section called “IndexWriteMaxOnDemandThroughputExceeded”

TableReadMaxOnDemandThroughputExceeded

Cuando esto ocurre

La tabla bajo demanda ha superado la capacidad de rendimiento de lectura máxima configurada. Puede supervisar las métricas de CloudWatch en the section called “Diagnóstico y supervisión comunes” para analizar el evento de limitación.

Opciones de corrección

Tenga en cuenta estos pasos para solucionar los problemas de limitación:

- Aumente el límite de rendimiento máximo: use la consola de DynamoDB, AWS CLI o la API de DynamoDB `UpdateTable` para aumentar el valor de `MaxReadRequestUnits` para la tabla afectada y, a continuación, supervise y ajuste. Esto permite que la tabla gestione un mayor rendimiento de lectura antes de que se produzca una limitación.
- Elimine el límite máximo: configure `MaxReadRequestUnits` en `-1` para eliminar el límite, lo que le permitirá escalar en función de la demanda hasta alcanzar las cuotas de rendimiento por cuenta. De esta forma, se elimina el límite personalizado, pero se mantienen las protecciones por cuenta de AWS. Sin embargo, es importante supervisar de cerca los gastos después de eliminar este

límite, ya que la tabla ahora puede consumir mucha más capacidad antes de alcanzar las cuotas por cuenta.

TableWriteMaxOnDemandThroughputExceeded

Cuando esto ocurre

La tabla bajo demanda ha superado la capacidad de rendimiento de escritura máxima configurada. Puede supervisar las métricas de CloudWatch en the section called “Diagnóstico y supervisión comunes” para analizar el evento de limitación.

Opciones de corrección

Tenga en cuenta estos pasos para solucionar los problemas de limitación:

- Aumente el límite de rendimiento máximo: use la consola de DynamoDB, AWS CLI o la API de DynamoDB `UpdateTable` para aumentar el valor de `MaxWriteRequestUnits` para la tabla afectada y, a continuación, supervise y ajuste.
- Elimine el límite máximo: configure `MaxWriteRequestUnits` en `-1` para eliminar el límite, lo que le permitirá escalar en función de la demanda hasta alcanzar las cuotas de rendimiento por cuenta. De esta forma, se elimina el límite personalizado, pero se mantienen las protecciones por cuenta de AWS. Sin embargo, es importante supervisar de cerca los gastos después de eliminar este límite, ya que la tabla ahora puede consumir mucha más capacidad antes de alcanzar las cuotas por cuenta.

IndexReadMaxOnDemandThroughputExceeded

Cuando esto ocurre

Las solicitudes de lectura a un GSI en modo bajo demanda han superado la capacidad de rendimiento máxima de lectura configurada para el GSI. Puede supervisar las métricas de CloudWatch en the section called “Diagnóstico y supervisión comunes” para analizar el evento de limitación.

Opciones de corrección

Tenga en cuenta estos pasos para solucionar los problemas de limitación:

- Aumente el límite de rendimiento máximo de GSI: use la consola de DynamoDB, AWS CLI o la API de DynamoDB `UpdateTable` para aumentar el valor de `MaxReadRequestUnits` para el GSI afectado y, a continuación, supervise y ajuste.
- Elimine el límite máximo de GSI: configure `MaxReadRequestUnits` en `-1` para el GSI para eliminar el límite, lo que le permitirá escalar en función de la demanda hasta alcanzar las cuotas de rendimiento por cuenta. De esta forma, se elimina el límite personalizado, pero se mantienen las protecciones por cuenta de AWS. Sin embargo, es importante supervisar los gastos de cerca después de eliminar este límite.

IndexWriteMaxOnDemandThroughputExceeded

Cuando esto ocurre

Las actualizaciones de los elementos de la tabla base desencadenan las escrituras en un GSI en modo bajo demanda que superan la capacidad de rendimiento de escritura máxima configurada por el GSI, lo que provoca una limitación de contrapresión. Puede supervisar las métricas de CloudWatch en the section called “Diagnóstico y supervisión comunes” para analizar el evento de limitación.

Opciones de corrección

Tenga en cuenta estos pasos para solucionar los problemas de limitación:

- Aumente el límite de rendimiento máximo de GSI: use la consola de DynamoDB, AWS CLI o la API de DynamoDB `UpdateTable` para aumentar el valor de `MaxWriteRequestUnits` para el GSI afectado y, a continuación, supervise y ajuste.
- Elimine el límite máximo de GSI: configure `MaxWriteRequestUnits` en `-1` para el GSI para eliminar el límite, lo que le permitirá escalar en función de la demanda hasta alcanzar las cuotas de rendimiento por cuenta. De esta forma, se elimina el límite personalizado, pero se mantienen las protecciones por cuenta de AWS. Sin embargo, es importante supervisar los gastos de cerca después de eliminar este límite.

Diagnóstico y supervisión comunes

Al solucionar problemas de superación del límite de rendimiento máximo bajo demanda, varias métricas de CloudWatch pueden ayudar a identificar la causa raíz y los patrones de escalado.

Métricas esenciales de CloudWatch

Supervise estas métricas de claves para diagnosticar la superación del límite de rendimiento máximo bajo demanda:

- Eventos de limitación de rendimiento máxima: `ReadMaxOnDemandThroughputThrottleEvents` y `WriteMaxOnDemandThroughputThrottleEvents` realizan un seguimiento cuando las solicitudes se limitan debido a la superación de los límites máximos. `ReadThrottleEvents` y `WriteThrottleEvents` realizan un seguimiento de los casos en que las solicitudes de lectura o escritura superan la capacidad aprovisionada.
- Rendimiento máximo actual configurado para una tabla o índice secundario global: `OnDemandMaxReadRequestUnits` y `OnDemandMaxWriteRequestUnits` muestran los límites de capacidad máxima actuales.
- Consumo de capacidad real: `ConsumedReadCapacityUnits` y `ConsumedWriteCapacityUnits` muestran los patrones de uso reales.

Enfoque de análisis

Siga estos pasos para confirmar que se ha superado el diagnóstico de rendimiento máximo bajo demanda:

1. Compare la capacidad consumida con los límites de capacidad máxima y compruebe si el consumo se acerca o supera de forma coherente los límites máximos.
2. Revise la frecuencia y el momento de los eventos de limitación para identificar los patrones. Esté atento a los aumentos repentinos de la capacidad consumida que coincidan con el evento de limitación.
3. Utilice Información·de·colaboradores·de·CloudWatch para identificar qué elementos o claves de partición consumen más capacidad.

Comprensión de la limitación de escritura del Índice Secundario Global (GSI) y la contrapresión en DynamoDB

La limitación de contrapresión de GSI representa uno de los escenarios de limitación más complejos de DynamoDB porque crea una relación indirecta entre las operaciones de escritura y la limitación: la aplicación escribe en una tabla base, pero sufre una limitación debido a las restricciones de capacidad en uno o varios índices.

Comprensión de la limitación por contrapresión de GSI

Cuando escribe en una tabla de DynamoDB, los índices secundarios globales (GSI) en esa tabla se actualizan de forma asincrónica mediante un modelo de consistencia final. Si un GSI no tiene la capacidad suficiente para gestionar estas actualizaciones, DynamoDB limita las escrituras en la tabla base para mantener la coherencia de datos. Esto se llama contrapresión de GSI. Para obtener más información sobre cómo funcionan los GSI, consulte Índices secundarios globales en DynamoDB.

A diferencia de la limitación directa de tablas, en la que el recurso al que se accede es también el que provoca la limitación, la contrapresión de GSI crea una dependencia entre la tabla base y sus índices. Incluso si la tabla base tiene una capacidad suficiente, las escrituras se limitarán si algún GSI asociado no puede gestionar el volumen de actualización. Es especialmente importante entender esta relación porque las restricciones por partición se aplican de forma independiente a la tabla base y a cada GSI; cada una tiene su propia estructura de particiones y los límites de rendimiento correspondientes.

La partición de GSI se basa en la clave de partición de GSI, que suele ser diferente de la clave de partición de la tabla base. Aunque el acceso a la tabla base esté perfectamente distribuido entre las particiones, es posible que las actualizaciones de GSI se concentren en particiones específicas, lo que generaría puntos calientes en el GSI. Para obtener información sobre las prácticas recomendadas generales sobre el diseño de claves de partición para tablas y para GSI, consulte Diseño de claves de partición de DynamoDB.

Por ejemplo, si la tabla base utiliza `customerId` como clave de partición (distribuida uniformemente) pero el GSI utiliza `status` como clave de partición (con valores posibles limitados, como “activo”, “pendiente” o “cerrado”), las actualizaciones de los elementos con valores de estado populares pueden crear particiones calientes de GSI incluso cuando el acceso a la tabla base esté equilibrado. Esto crea un escenario particularmente desafiante en el que la aplicación podría sufrir una limitación debido a las particiones calientes de GSI, aunque la tabla base y el GSI tengan suficiente capacidad general y el patrón de acceso de la tabla base parezca estar bien distribuido.

Aunque la excepción de limitación apunta al GSI (a través de `ResourceArn`), la operación que en realidad se está limitando es la escritura en la tabla base. Esto puede resultar confuso porque la aplicación escribe en la tabla base, pero recibe una excepción relacionada con el GSI.

Tipos de limitación del GSI

La limitación de la contrapresión de GSI se manifiesta mediante diferentes tipos de excepciones en función de la restricción de capacidad específica:

- Se ha superado la capacidad aprovisionada de GSI: se produce cuando el GSI carece de unidades de capacidad de escritura suficientes para gestionar las actualizaciones de las operaciones de la tabla base. Esto produce una `ProvisionedThroughputExceededException` con el motivo `IndexWriteProvisionedThroughputExceeded` y el `ResourceArn` apunta directamente al GSI específico que está experimentando restricciones de capacidad.
- Se ha superado el rendimiento máximo bajo demanda de GSI: se produce cuando las operaciones de escritura de GSI superan los límites máximos configurados en las tablas bajo demanda. Esto produce una `ThrottlingException` con el motivo `IndexWriteMaxOnDemandThroughputExceeded`, lo que identifica el GSI específico con las restricciones de rendimiento configuradas.
- Se han superado los límites de partición de GSI: se produce cuando las particiones de GSI individuales superan sus límites de rendimiento (particiones calientes), incluso si la capacidad general del GSI parece suficiente. Esto genera una `ThrottlingException` con el motivo `IndexWriteKeyRangeThroughputExceeded`, que indica problemas de particiones calientes en el GSI específico identificado en el `ResourceArn`. Esto es particularmente importante porque la distribución de particiones del GSI puede diferir significativamente de la distribución de particiones de la tabla base, lo que crea puntos calientes en el GSI incluso cuando el acceso a la tabla base se distribuye de manera uniforme.
- Se han superado los límites de las cuentas de GSI: se desencadenan cuando las operaciones de escritura en un GSI específico superan los límites de rendimiento regional por tabla (o cualquier GSI individual de esa tabla) establecidos por cuenta. DynamoDB devuelve una `ThrottlingException` con el motivo `IndexWriteAccountLimitExceeded`, que apunta al GSI que llevó su uso más allá de los límites de la cuenta. Esta limitación se produce de forma independiente para cada GSI que supera el límite. Para obtener información sobre las cuotas por tabla, por cuenta, regionales y de servicio, consulte the section called “Cuotas”.

Métricas de limitación de CloudWatch

Esta página proporciona una guía completa de las métricas de CloudWatch diseñada específicamente para ayudarle a identificar, diagnosticar y resolver los problemas de limitación en las tablas e índices de DynamoDB.

Métricas generales de limitación

- `ThrottledRequests`
  - Se incrementa en uno cuando se limita cualquier evento dentro de una solicitud, independientemente del número de eventos individuales de esa solicitud limitados. Por

ejemplo, cuando se actualiza un elemento de una tabla con índices secundarios globales (GSI), se producen varios eventos: una operación de escritura en la tabla base y una operación de escritura en cada índice. Si alguno de estos eventos individuales está limitado, la métrica `ThrottledRequests` solo se incrementa una vez.

Es importante entender este comportamiento a la hora de supervisar y solucionar problemas del rendimiento de DynamoDB, ya que puede ocultar el verdadero alcance de la limitación. Para obtener información más completa, compare la métrica `ThrottledRequests` con las métricas específicas por evento, como `ReadThrottleEvents`, `WriteThrottleEvents`, y con las métricas específicas como `ReadKeyRangeThroughputThrottleEvents`, por ejemplo. La lista completa de estas métricas por causas específicas está disponible en esta página. Cada métrica corresponde a motivos de limitación específicos que se incluyen en la excepción de limitación. Para obtener instrucciones sobre cómo recuperar e interpretar estos motivos durante los eventos de limitación, consulte la sección the section called “Diagnóstico de limitación” que proporciona instrucciones para identificar y resolver las causas fundamentales de los problemas de limitación.

- `ReadThrottleEvents`
  - Preste atención a las solicitudes que superen la RCU aprovisionada para una tabla o GSI.
- `WriteThrottleEvents`
  - Preste atención a las solicitudes que superen la WCU aprovisionada para una tabla o GSI.

Métricas de limitación detalladas por causa

Limitación del rendimiento bajo demanda

- `ReadMaxOnDemandThroughputThrottleEvents`
  - El número de solicitudes de lectura se ha limitado debido al rendimiento máximo bajo demanda.
- `WriteMaxOnDemandThroughputThrottleEvents`
  - El número de solicitudes de escritura se ha limitado debido al rendimiento máximo bajo demanda.

Limitaciones por cuenta

- `ReadAccountLimitThrottleEvents`
  - Número de solicitudes de lectura limitadas debido a los límites de la cuenta.
- `WriteAccountLimitThrottleEvents`

- Número de solicitudes de escritura limitadas debido a los límites de la cuenta.

Limitaciones por partición

- `ReadKeyRangeThroughputThrottleEvents`
  - Número de solicitudes de lectura limitadas debido a los límites de las particiones.
- `WriteKeyRangeThroughputThrottleEvents`
  - Número de solicitudes de escritura limitadas debido a los límites de las particiones.

Métricas de análisis de capacidad

- `OnlineIndexConsumedWriteCapacity`
  - Al agregar un GSI nuevo a una tabla existente, DynamoDB realiza una operación de relleno que copia los datos de la tabla base al nuevo índice. Este proceso consume unidades de capacidad de escritura. La métrica `OnlineIndexConsumedWriteCapacity` registra este consumo específico.

    Este consumo es independiente y adicional a las operaciones de escritura habituales rastreadas por `ConsumedWriteCapacityUnits`. La métrica `ConsumedWriteCapacityUnits` regular para un GSI no incluye la capacidad de escritura consumida durante el proceso de creación del índice inicial.
- `ProvisionedReadCapacityUnits` y `ProvisionedWriteCapacityUnits`
  - Vea cuántas unidades de capacidad de lectura o escritura aprovisionadas se han consumido durante el periodo de tiempo especificado, para una tabla o un índice secundario global especificado.
  - Tenga en cuenta que la dimensión `TableName` devuelve `ProvisionedReadCapacityUnits` para la tabla solo de forma predeterminada. Para ver el número de unidades de capacidad de lectura o escritura aprovisionadas para un índice secundario global, debe especificar `TableName` y `GlobalSecondaryIndexName`.
- `ConsumedReadCapacityUnits` y `ConsumedWriteCapacityUnits`
  - Consulte cuántas unidades de capacidad de lectura o escritura se han consumido durante el periodo de tiempo especificado. `ConsumedWriteCapacityUnits` no incluye la capacidad de escritura consumida durante el proceso inicial de creación del índice.

Para obtener más información sobre las métricas de DynamoDB CloudWatch, consulte Dimensiones y métricas de DynamoDB.

Apéndice DynamoDB

Temas

- Solución de problemas de establecimiento de conexiones SSL/TLS con DynamoDB
- Tablas y datos de ejemplo para usar en DynamoDB
- Creación de tablas de ejemplo y carga de datos en DynamoDB
- Aplicación de ejemplo de DynamoDB con AWS SDK for Python (Boto): Tic-Tac-Toe (Tres en raya)
- Palabras reservadas en DynamoDB
- AWSEjemplos de SDK de para Java 1.x
- Ejemplos de AWS SDK para Go 1.x
- Ejemplos de AWS SDK para Node.js 2.x

Solución de problemas de establecimiento de conexiones SSL/TLS con DynamoDB

Amazon DynamoDB está moviendo nuestros puntos de enlace a certificados seguros firmados por la entidad de certificación de Amazon Trust Services (ATS) en lugar de una entidad de certificación de terceros. En diciembre de 2017, lanzamos la región EU-WEST-3 (París) con los certificados seguros emitidos por Amazon Trust Services. Todas las nuevas regiones lanzadas después de diciembre de 2017 tienen puntos de enlace con los certificados emitidos por Amazon Trust Services. Esta guía le muestra cómo validar y solucionar los problemas de conexión SSL/TLS.

Probar su aplicación o servicio

La mayoría de los SDK de AWS y las interfaces de línea de comandos (CLI) admiten la entidad de certificación de Amazon Trust Services. Si está utilizando una versión de SDK de AWS para Python o CLI publicado antes del 29 de octubre de 2013, debe actualizarlo. Los SDK y CLI de .NET, Java, PHP, Go, JavaScript y C++ no agrupan ningún certificado, sus certificados provienen del sistema operativo subyacente. El SDK de Ruby ha incluido al menos una de las entidades de certificación requeridas desde el 10 de junio de 2015. Antes de esa fecha, el SDK de Ruby V2 no agrupaba certificados. Si utiliza una versión no compatible, personalizada o modificada de los SDK de AWS, o si utiliza un almacén de confianza personalizado, es posible que no cuente con el soporte necesario para la entidad de certificación de Amazon Trust Services.

Para validar el acceso a los puntos de enlace de DynamoDB, deberá desarrollar una prueba que acceda a la API de DynamoDB o a la API de DynamoDB Streams en la región EU-WEST-3 y validar que el establecimiento de comunicación TLS tenga éxito. Los puntos de enlace específicos a los que deberá acceder en dicha prueba son:

- DynamoDB: https://dynamodb.eu-west-3.amazonaws.com
- DynamoDB Streams: https://streams.dynamodb.eu-west-3.amazonaws.com

Si su aplicación no es compatible con la entidad de certificación de Amazon Trust Services, verá uno de los siguientes errores:

- Errores de negociación SSL/TLS
- Un largo retraso antes de que el software reciba un error que indica la falla de negociación SSL/TLS. El tiempo de retraso depende de la estrategia de reintento y la configuración del tiempo de espera de su cliente.

Probar el navegador del cliente

Para comprobar que su navegador puede conectarse a Amazon DynamoDB, abra la siguiente dirección URL: https://dynamodb.eu-west-3.amazonaws.com. Si la prueba se realiza correctamente, aparecerá un mensaje como este:

```
healthy: dynamodb.eu-west-3.amazonaws.com
```

Si la prueba no tiene éxito, mostrará un error similar al siguiente: https://untrusted-root.badssl.com/.

Actualización del cliente de aplicación de software

Las aplicaciones que obtengan acceso a los puntos de enlace de la API de DynamoDB o DynamoDB Streams (ya sea a través de un navegador o mediante programación) deberán actualizar la lista de entidades de certificación de confianza en los equipos del cliente si aún no admiten alguna de las siguientes entidades de certificación:

- Amazon Root CA 1
- Starfield Services Root Certificate Authority - G2
- Starfield Class 2 Certification Authority

Si los clientes ya confían en CUALQUIERA de las tres entidades de certificaciones anteriores, estas confiarán en los certificados utilizados por DynamoDB y no se requiere ninguna acción. Sin embargo, si sus clientes no confían en ninguna de las entidades de certificación anteriores, las conexiones HTTPS a las API de DynamoDB o DynamoDB Streams van a fallar. Para obtener más información, visite esta entrada al blog: https://aws.amazon.com/blogs/security/how-to-prepare-for-aws-move-to-its-own-certificate-authority/.

Actualización del navegador del cliente

Puede actualizar el paquete de certificados en su navegador simplemente actualizando su navegador. Las instrucciones para los navegadores más comunes se pueden encontrar en los sitios web de los navegadores:

- Chrome: https://support.google.com/chrome/answer/95414?hl=en
- Firefox: https://support.mozilla.org/en-US/kb/update-firefox-latest-version
- Safari: https://support.apple.com/en-us/HT204416
- Internet Explorer: https://support.microsoft.com/en-us/help/17295/windows-internet-explorer-which-version#ie=other

Actualización manual su paquete de certificados

Si no puede acceder a la API de DynamoDB o la API de DynamoDB Streams, entonces deberá actualizar la agrupación de certificados. Para ello, debe importar al menos una de las entidades de certificación requeridas. Puede encontrarlas en https://www.amazontrust.com/repository/.

Los siguientes sistemas operativos y lenguajes de programación admiten certificados de Amazon Trust Services:

- Versiones de Microsoft Windows que tienen instaladas actualizaciones de enero de 2005 o posteriores, Windows Vista, Windows 7, Windows Server 2008 y versiones más recientes.
- MacOS X 10.4 con Java para MacOS X 10.4 versión 5, MacOS X 10.5 y versiones más recientes.
- Red Hat Enterprise Linux 5 (marzo de 2007), Linux 6 y Linux 7 y CentOS 5, CentOS 6 y CentOS 7
- Ubuntu 8.10
- Debian 5.0
- Amazon Linux (todas las versiones)

- Java 1.4.2_12, Java 5 actualización 2 y todas las versiones más recientes, incluyendo Java 6, Java 7 y Java 8

Si aún no puede conectarse, consulte la documentación del software, el proveedor del sistema operativo o contacte con AWS Support https://aws.amazon.com/support para obtener más ayuda.

Tablas y datos de ejemplo para usar en DynamoDB

La Guía para desarrolladores de Amazon DynamoDB utiliza ejemplos de tablas para ilustrar diversos aspectos de DynamoDB.

Nombre de la tabla	Clave principal
ProductCatalog	Clave principal simple: • `Id` (Número)
Forum	Clave principal simple: • `Name` (Cadena)
Thread	Clave principal compuesta: • `ForumName` (Cadena) • `Subject` (Cadena)
Responder	Clave principal compuesta: • `Id` (Cadena) • `ReplyDateTime` (Cadena)

La tabla Reply posee un índice secundario global denominado PostedBy-Message-Index. Este índice facilitará las consultas de dos atributos sin clave de la tabla Reply.

Nombre de índice	Clave principal
PostedBy-Message-Index	Clave principal compuesta:

Nombre de índice	Clave principal
	• `PostedBy` (Cadena) • `Message` (Cadena)

Para obtener más información sobre estas tablas, consulte Paso 1: crear una tabla en DynamoDB y Paso 2: escribir datos en una tabla de DynamoDB.

Ejemplos de archivos de datos

Temas

- Ejemplos de datos de ProductCatalog
- Ejemplo de datos de Forum
- Ejemplo de datos de Thread
- Ejemplo de datos de Reply

En las secciones siguientes se muestran los ejemplos de archivos de datos que se utilizan para cargar las tablas ProductCatalog, Forum, Thread y Reply.

Cada archivo de datos contiene varios componentes `PutRequest`, cada uno de los cuales contienen un único elemento. Estos componentes `PutRequest` se utilizan como datos de entrada de la operación `BatchWriteItem`, mediante la AWS Command Line Interface (AWS CLI).

Ejemplos de datos de ProductCatalog

```
{
    "ProductCatalog": [
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "101"
                    },
                    "Title": {
                        "S": "Book 101 Title"
                    },
                    "ISBN": {
                        "S": "111-1111111111"
```

```
                },
                "Authors": {
                    "L": [
                        {
                            "S": "Author1"
                        }
                    ]
                },
                "Price": {
                    "N": "2"
                },
                "Dimensions": {
                    "S": "8.5 x 11.0 x 0.5"
                },
                "PageCount": {
                    "N": "500"
                },
                "InPublication": {
                    "BOOL": true
                },
                "ProductCategory": {
                    "S": "Book"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "102"
                },
                "Title": {
                    "S": "Book 102 Title"
                },
                "ISBN": {
                    "S": "222-2222222222"
                },
                "Authors": {
                    "L": [
                        {
                            "S": "Author1"
                        },
                        {
```

```
                            "S": "Author2"
                        }
                    ]
                },
                "Price": {
                    "N": "20"
                },
                "Dimensions": {
                    "S": "8.5 x 11.0 x 0.8"
                },
                "PageCount": {
                    "N": "600"
                },
                "InPublication": {
                    "BOOL": true
                },
                "ProductCategory": {
                    "S": "Book"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "103"
                },
                "Title": {
                    "S": "Book 103 Title"
                },
                "ISBN": {
                    "S": "333-3333333333"
                },
                "Authors": {
                    "L": [
                        {
                            "S": "Author1"
                        },
                        {
                            "S": "Author2"
                        }
                    ]
                },
```

```
                "Price": {
                    "N": "2000"
                },
                "Dimensions": {
                    "S": "8.5 x 11.0 x 1.5"
                },
                "PageCount": {
                    "N": "600"
                },
                "InPublication": {
                    "BOOL": false
                },
                "ProductCategory": {
                    "S": "Book"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "201"
                },
                "Title": {
                    "S": "18-Bike-201"
                },
                "Description": {
                    "S": "201 Description"
                },
                "BicycleType": {
                    "S": "Road"
                },
                "Brand": {
                    "S": "Mountain A"
                },
                "Price": {
                    "N": "100"
                },
                "Color": {
                    "L": [
                        {
                            "S": "Red"
                        },
```

```
                        {
                            "S": "Black"
                        }
                    ]
                },
                "ProductCategory": {
                    "S": "Bicycle"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "202"
                },
                "Title": {
                    "S": "21-Bike-202"
                },
                "Description": {
                    "S": "202 Description"
                },
                "BicycleType": {
                    "S": "Road"
                },
                "Brand": {
                    "S": "Brand-Company A"
                },
                "Price": {
                    "N": "200"
                },
                "Color": {
                    "L": [
                        {
                            "S": "Green"
                        },
                        {
                            "S": "Black"
                        }
                    ]
                },
                "ProductCategory": {
                    "S": "Bicycle"
```

```
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "N": "203"
                },
                "Title": {
                    "S": "19-Bike-203"
                },
                "Description": {
                    "S": "203 Description"
                },
                "BicycleType": {
                    "S": "Road"
                },
                "Brand": {
                    "S": "Brand-Company B"
                },
                "Price": {
                    "N": "300"
                },
                "Color": {
                    "L": [
                        {
                            "S": "Red"
                        },
                        {
                            "S": "Green"
                        },
                        {
                            "S": "Black"
                        }
                    ]
                },
                "ProductCategory": {
                    "S": "Bicycle"
                }
            }
        }
    },
```

```
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "204"
                    },
                    "Title": {
                        "S": "18-Bike-204"
                    },
                    "Description": {
                        "S": "204 Description"
                    },
                    "BicycleType": {
                        "S": "Mountain"
                    },
                    "Brand": {
                        "S": "Brand-Company B"
                    },
                    "Price": {
                        "N": "400"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Red"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "N": "205"
                    },
                    "Title": {
                        "S": "18-Bike-204"
                    },
                    "Description": {
```

```
                        "S": "205 Description"
                    },
                    "BicycleType": {
                        "S": "Hybrid"
                    },
                    "Brand": {
                        "S": "Brand-Company C"
                    },
                    "Price": {
                        "N": "500"
                    },
                    "Color": {
                        "L": [
                            {
                                "S": "Red"
                            },
                            {
                                "S": "Black"
                            }
                        ]
                    },
                    "ProductCategory": {
                        "S": "Bicycle"
                    }
                }
            }
        }
    ]
}
```

Ejemplo de datos de Forum

```
{
    "Forum": [
        {
            "PutRequest": {
                "Item": {
                    "Name": {"S":"Amazon DynamoDB"},
                    "Category": {"S":"Amazon Web Services"},
                    "Threads": {"N":"2"},
                    "Messages": {"N":"4"},
                    "Views": {"N":"1000"}
                }
```

```
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Name": {"S":"Amazon S3"},
                    "Category": {"S":"Amazon Web Services"}
                }
            }
        }
    ]
}
```

Ejemplo de datos de Thread

```
{
    "Thread": [
        {
            "PutRequest": {
                "Item": {
                    "ForumName": {
                        "S": "Amazon DynamoDB"
                    },
                    "Subject": {
                        "S": "DynamoDB Thread 1"
                    },
                    "Message": {
                        "S": "DynamoDB thread 1 message"
                    },
                    "LastPostedBy": {
                        "S": "User A"
                    },
                    "LastPostedDateTime": {
                        "S": "2015-09-22T19:58:22.514Z"
                    },
                    "Views": {
                        "N": "0"
                    },
                    "Replies": {
                        "N": "0"
                    },
                    "Answered": {
                        "N": "0"
```

```
                },
                "Tags": {
                    "L": [
                        {
                            "S": "index"
                        },
                        {
                            "S": "primarykey"
                        },
                        {
                            "S": "table"
                        }
                    ]
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "ForumName": {
                    "S": "Amazon DynamoDB"
                },
                "Subject": {
                    "S": "DynamoDB Thread 2"
                },
                "Message": {
                    "S": "DynamoDB thread 2 message"
                },
                "LastPostedBy": {
                    "S": "User A"
                },
                "LastPostedDateTime": {
                    "S": "2015-09-15T19:58:22.514Z"
                },
                "Views": {
                    "N": "3"
                },
                "Replies": {
                    "N": "0"
                },
                "Answered": {
                    "N": "0"
                },
```

```
                "Tags": {
                    "L": [
                        {
                            "S": "items"
                        },
                        {
                            "S": "attributes"
                        },
                        {
                            "S": "throughput"
                        }
                    ]
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "ForumName": {
                    "S": "Amazon S3"
                },
                "Subject": {
                    "S": "S3 Thread 1"
                },
                "Message": {
                    "S": "S3 thread 1 message"
                },
                "LastPostedBy": {
                    "S": "User A"
                },
                "LastPostedDateTime": {
                    "S": "2015-09-29T19:58:22.514Z"
                },
                "Views": {
                    "N": "0"
                },
                "Replies": {
                    "N": "0"
                },
                "Answered": {
                    "N": "0"
                },
                "Tags": {
```

```
                        "L": [
                            {
                                "S": "largeobjects"
                            },
                            {
                                "S": "multipart upload"
                            }
                        ]
                    }
                }
            }
        }
    ]
}
```

Ejemplo de datos de Reply

```
{
    "Reply": [
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 1"
                    },
                    "ReplyDateTime": {
                        "S": "2015-09-15T19:58:22.947Z"
                    },
                    "Message": {
                        "S": "DynamoDB Thread 1 Reply 1 text"
                    },
                    "PostedBy": {
                        "S": "User A"
                    }
                }
            }
        },
        {
            "PutRequest": {
                "Item": {
                    "Id": {
                        "S": "Amazon DynamoDB#DynamoDB Thread 1"
                    },
```

```
                "ReplyDateTime": {
                    "S": "2015-09-22T19:58:22.947Z"
                },
                "Message": {
                    "S": "DynamoDB Thread 1 Reply 2 text"
                },
                "PostedBy": {
                    "S": "User B"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "S": "Amazon DynamoDB#DynamoDB Thread 2"
                },
                "ReplyDateTime": {
                    "S": "2015-09-29T19:58:22.947Z"
                },
                "Message": {
                    "S": "DynamoDB Thread 2 Reply 1 text"
                },
                "PostedBy": {
                    "S": "User A"
                }
            }
        }
    },
    {
        "PutRequest": {
            "Item": {
                "Id": {
                    "S": "Amazon DynamoDB#DynamoDB Thread 2"
                },
                "ReplyDateTime": {
                    "S": "2015-10-05T19:58:22.947Z"
                },
                "Message": {
                    "S": "DynamoDB Thread 2 Reply 2 text"
                },
                "PostedBy": {
                    "S": "User A"
```

```
                    }
                }
            }
        }
    ]
}
```

Creación de tablas de ejemplo y carga de datos en DynamoDB

En este apéndice se proporciona código para crear las tablas y añadirles datos mediante programación.

Temas

- Creación de ejemplos de tablas y carga de datos utilizando AWS SDK para Java
- Creación de ejemplos de tablas y carga de datos utilizando AWS SDK para .NET

Creación de ejemplos de tablas y carga de datos utilizando AWS SDK para Java

En el siguiente ejemplo de código Java se crean tablas y se cargan datos en ellas. Para obtener instrucciones paso a paso sobre cómo ejecutar este código mediante Eclipse, consulte Ejemplos de código Java.

```
package com.amazonaws.codesamples;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.HashSet;
import java.util.TimeZone;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.CreateTableRequest;
```

```
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.LocalSecondaryIndex;
import com.amazonaws.services.dynamodbv2.model.Projection;
import com.amazonaws.services.dynamodbv2.model.ProjectionType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;

public class CreateTablesLoadData {

    static AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
    static DynamoDB dynamoDB = new DynamoDB(client);

    static SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'");

    static String productCatalogTableName = "ProductCatalog";
    static String forumTableName = "Forum";
    static String threadTableName = "Thread";
    static String replyTableName = "Reply";

    public static void main(String[] args) throws Exception {

        try {

            deleteTable(productCatalogTableName);
            deleteTable(forumTableName);
            deleteTable(threadTableName);
            deleteTable(replyTableName);

            // Parameter1: table name
            // Parameter2: reads per second
            // Parameter3: writes per second
            // Parameter4/5: partition key and data type
            // Parameter6/7: sort key and data type (if applicable)

            createTable(productCatalogTableName, 10L, 5L, "Id", "N");
            createTable(forumTableName, 10L, 5L, "Name", "S");
            createTable(threadTableName, 10L, 5L, "ForumName", "S", "Subject", "S");
            createTable(replyTableName, 10L, 5L, "Id", "S", "ReplyDateTime", "S");

            loadSampleProducts(productCatalogTableName);
            loadSampleForums(forumTableName);
            loadSampleThreads(threadTableName);
            loadSampleReplies(replyTableName);
```

```
        } catch (Exception e) {
            System.err.println("Program failed:");
            System.err.println(e.getMessage());
        }
        System.out.println("Success.");
    }

    private static void deleteTable(String tableName) {
        Table table = dynamoDB.getTable(tableName);
        try {
            System.out.println("Issuing DeleteTable request for " + tableName);
            table.delete();
            System.out.println("Waiting for " + tableName + " to be deleted...this may
take a while...");
            table.waitForDelete();

        } catch (Exception e) {
            System.err.println("DeleteTable request failed for " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void createTable(String tableName, long readCapacityUnits, long
writeCapacityUnits,
            String partitionKeyName, String partitionKeyType) {

        createTable(tableName, readCapacityUnits, writeCapacityUnits, partitionKeyName,
partitionKeyType, null, null);
    }

    private static void createTable(String tableName, long readCapacityUnits, long
writeCapacityUnits,
            String partitionKeyName, String partitionKeyType, String sortKeyName,
String sortKeyType) {

        try {

            ArrayList<KeySchemaElement> keySchema = new ArrayList<KeySchemaElement>();
            keySchema.add(new
KeySchemaElement().withAttributeName(partitionKeyName).withKeyType(KeyType.HASH)); //
Partition

                            // key
```

```
            ArrayList<AttributeDefinition> attributeDefinitions = new
ArrayList<AttributeDefinition>();
            attributeDefinitions
                    .add(new AttributeDefinition().withAttributeName(partitionKeyName)
                            .withAttributeType(partitionKeyType));

            if (sortKeyName != null) {
                keySchema.add(new
KeySchemaElement().withAttributeName(sortKeyName).withKeyType(KeyType.RANGE)); // Sort

                              // key
                attributeDefinitions
                        .add(new
AttributeDefinition().withAttributeName(sortKeyName).withAttributeType(sortKeyType));
            }

            CreateTableRequest request = new
CreateTableRequest().withTableName(tableName).withKeySchema(keySchema)
                    .withProvisionedThroughput(new
ProvisionedThroughput().withReadCapacityUnits(readCapacityUnits)
                            .withWriteCapacityUnits(writeCapacityUnits));

            // If this is the Reply table, define a local secondary index
            if (replyTableName.equals(tableName)) {

                attributeDefinitions
                        .add(new
AttributeDefinition().withAttributeName("PostedBy").withAttributeType("S"));

                ArrayList<LocalSecondaryIndex> localSecondaryIndexes = new
ArrayList<LocalSecondaryIndex>();
                localSecondaryIndexes.add(new
LocalSecondaryIndex().withIndexName("PostedBy-Index")
                        .withKeySchema(
                                new
KeySchemaElement().withAttributeName(partitionKeyName).withKeyType(KeyType.HASH), //
Partition

                               // key
                                new
KeySchemaElement().withAttributeName("PostedBy").withKeyType(KeyType.RANGE)) // Sort

                              // key
```

```
                        .withProjection(new
Projection().withProjectionType(ProjectionType.KEYS_ONLY)));

                request.setLocalSecondaryIndexes(localSecondaryIndexes);
            }

            request.setAttributeDefinitions(attributeDefinitions);

            System.out.println("Issuing CreateTable request for " + tableName);
            Table table = dynamoDB.createTable(request);
            System.out.println("Waiting for " + tableName + " to be created...this may
take a while...");
            table.waitForActive();

        } catch (Exception e) {
            System.err.println("CreateTable request failed for " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void loadSampleProducts(String tableName) {

        Table table = dynamoDB.getTable(tableName);

        try {

            System.out.println("Adding data to " + tableName);

            Item item = new Item().withPrimaryKey("Id", 101).withString("Title", "Book
101 Title")
                    .withString("ISBN", "111-1111111111")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author1"))).withNumber("Price", 2)
                    .withString("Dimensions", "8.5 x 11.0 x
0.5").withNumber("PageCount", 500)
                    .withBoolean("InPublication", true).withString("ProductCategory",
"Book");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 102).withString("Title", "Book 102
Title")
                    .withString("ISBN", "222-2222222222")
                    .withStringSet("Authors", new
HashSet<String>(Arrays.asList("Author1", "Author2")))
```

```
                    .withNumber("Price", 20).withString("Dimensions", "8.5 x 11.0 x
 0.8").withNumber("PageCount", 600)
                    .withBoolean("InPublication", true).withString("ProductCategory",
 "Book");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 103).withString("Title", "Book 103
 Title")
                    .withString("ISBN", "333-3333333333")
                    .withStringSet("Authors", new
 HashSet<String>(Arrays.asList("Author1", "Author2")))
                    // Intentional. Later we'll run Scan to find price error. Find
                    // items > 1000 in price.
                    .withNumber("Price", 2000).withString("Dimensions", "8.5 x 11.0 x
 1.5").withNumber("PageCount", 600)
                    .withBoolean("InPublication", false).withString("ProductCategory",
 "Book");
            table.putItem(item);

            // Add bikes.

            item = new Item().withPrimaryKey("Id", 201).withString("Title", "18-
Bike-201")
                    // Size, followed by some title.
                    .withString("Description", "201
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Mountain A")
                    // Trek, Specialized.
                    .withNumber("Price", 100).withStringSet("Color", new
 HashSet<String>(Arrays.asList("Red", "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 202).withString("Title", "21-
Bike-202")
                    .withString("Description", "202
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Brand-Company A").withNumber("Price", 200)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Green",
 "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);
```

```
            item = new Item().withPrimaryKey("Id", 203).withString("Title", "19-
Bike-203")
                    .withString("Description", "203
 Description").withString("BicycleType", "Road")
                    .withString("Brand", "Brand-Company B").withNumber("Price", 300)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red",
 "Green", "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 204).withString("Title", "18-
Bike-204")
                    .withString("Description", "204
 Description").withString("BicycleType", "Mountain")
                    .withString("Brand", "Brand-Company B").withNumber("Price", 400)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", 205).withString("Title", "20-
Bike-205")
                    .withString("Description", "205
 Description").withString("BicycleType", "Hybrid")
                    .withString("Brand", "Brand-Company C").withNumber("Price", 500)
                    .withStringSet("Color", new HashSet<String>(Arrays.asList("Red",
 "Black")))
                    .withString("ProductCategory", "Bicycle");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }

    }

    private static void loadSampleForums(String tableName) {

        Table table = dynamoDB.getTable(tableName);

        try {

            System.out.println("Adding data to " + tableName);
```

```
            Item item = new Item().withPrimaryKey("Name", "Amazon DynamoDB")
                    .withString("Category", "Amazon Web
Services").withNumber("Threads", 2).withNumber("Messages", 4)
                    .withNumber("Views", 1000);
            table.putItem(item);

            item = new Item().withPrimaryKey("Name", "Amazon
S3").withString("Category", "Amazon Web Services")
                    .withNumber("Threads", 0);
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }
    }

    private static void loadSampleThreads(String tableName) {
        try {
            long time1 = (new Date()).getTime() - (7 * 24 * 60 * 60 * 1000); // 7
            // days
            // ago
            long time2 = (new Date()).getTime() - (14 * 24 * 60 * 60 * 1000); // 14
            // days
            // ago
            long time3 = (new Date()).getTime() - (21 * 24 * 60 * 60 * 1000); // 21
            // days
            // ago

            Date date1 = new Date();
            date1.setTime(time1);

            Date date2 = new Date();
            date2.setTime(time2);

            Date date3 = new Date();
            date3.setTime(time3);

            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));

            Table table = dynamoDB.getTable(tableName);

            System.out.println("Adding data to " + tableName);
```

```
            Item item = new Item().withPrimaryKey("ForumName", "Amazon DynamoDB")
                    .withString("Subject", "DynamoDB Thread 1").withString("Message",
"DynamoDB thread 1 message")
                    .withString("LastPostedBy", "User
A").withString("LastPostedDateTime", dateFormatter.format(date2))
                    .withNumber("Views", 0).withNumber("Replies",
0).withNumber("Answered", 0)
                    .withStringSet("Tags", new HashSet<String>(Arrays.asList("index",
"primarykey", "table")));
            table.putItem(item);

            item = new Item().withPrimaryKey("ForumName", "Amazon
DynamoDB").withString("Subject", "DynamoDB Thread 2")
                    .withString("Message", "DynamoDB thread 2
message").withString("LastPostedBy", "User A")
                    .withString("LastPostedDateTime",
dateFormatter.format(date3)).withNumber("Views", 0)
                    .withNumber("Replies", 0).withNumber("Answered", 0)
                    .withStringSet("Tags", new HashSet<String>(Arrays.asList("index",
"partitionkey", "sortkey")));
            table.putItem(item);

            item = new Item().withPrimaryKey("ForumName", "Amazon
S3").withString("Subject", "S3 Thread 1")
                    .withString("Message", "S3 Thread 3
message").withString("LastPostedBy", "User A")
                    .withString("LastPostedDateTime",
dateFormatter.format(date1)).withNumber("Views", 0)
                    .withNumber("Replies", 0).withNumber("Answered", 0)
                    .withStringSet("Tags", new
HashSet<String>(Arrays.asList("largeobjects", "multipart upload")));
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());
        }

    }

    private static void loadSampleReplies(String tableName) {
        try {
            // 1 day ago
            long time0 = (new Date()).getTime() - (1 * 24 * 60 * 60 * 1000);
```

```
            // 7 days ago
            long time1 = (new Date()).getTime() - (7 * 24 * 60 * 60 * 1000);
            // 14 days ago
            long time2 = (new Date()).getTime() - (14 * 24 * 60 * 60 * 1000);
            // 21 days ago
            long time3 = (new Date()).getTime() - (21 * 24 * 60 * 60 * 1000);

            Date date0 = new Date();
            date0.setTime(time0);

            Date date1 = new Date();
            date1.setTime(time1);

            Date date2 = new Date();
            date2.setTime(time2);

            Date date3 = new Date();
            date3.setTime(time3);

            dateFormatter.setTimeZone(TimeZone.getTimeZone("UTC"));

            Table table = dynamoDB.getTable(tableName);

            System.out.println("Adding data to " + tableName);

            // Add threads.

            Item item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB
Thread 1")
                    .withString("ReplyDateTime", (dateFormatter.format(date3)))
                    .withString("Message", "DynamoDB Thread 1 Reply 1
text").withString("PostedBy", "User A");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 1")
                    .withString("ReplyDateTime", dateFormatter.format(date2))
                    .withString("Message", "DynamoDB Thread 1 Reply 2
text").withString("PostedBy", "User B");
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 2")
                    .withString("ReplyDateTime", dateFormatter.format(date1))
                    .withString("Message", "DynamoDB Thread 2 Reply 1
text").withString("PostedBy", "User A");
```

```
            table.putItem(item);

            item = new Item().withPrimaryKey("Id", "Amazon DynamoDB#DynamoDB Thread 2")
                    .withString("ReplyDateTime", dateFormatter.format(date0))
                    .withString("Message", "DynamoDB Thread 2 Reply 2
 text").withString("PostedBy", "User A");
            table.putItem(item);

        } catch (Exception e) {
            System.err.println("Failed to create item in " + tableName);
            System.err.println(e.getMessage());

        }
    }

}
```

Creación de ejemplos de tablas y carga de datos utilizando AWS SDK para .NET

En el siguiente ejemplo de código C# se crean tablas y se cargan datos en ellas. Para obtener instrucciones paso a paso sobre cómo ejecutar este código en Visual Studio, consulte Ejemplos de código .NET.

```
using System;
using System.Collections.Generic;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;
using Amazon.Runtime;
using Amazon.SecurityToken;

namespace com.amazonaws.codesamples
{
    class CreateTablesLoadData
    {
        private static AmazonDynamoDBClient client = new AmazonDynamoDBClient();

        static void Main(string[] args)
        {
            try
```

```
            {
                //DeleteAllTables(client);
                DeleteTable("ProductCatalog");
                DeleteTable("Forum");
                DeleteTable("Thread");
                DeleteTable("Reply");

                // Create tables (using the AWS SDK for .NET low-level API).
                CreateTableProductCatalog();
                CreateTableForum();
                CreateTableThread(); // ForumTitle, Subject */
                CreateTableReply();

                // Load data (using the .NET SDK document API)
                LoadSampleProducts();
                LoadSampleForums();
                LoadSampleThreads();
                LoadSampleReplies();
                Console.WriteLine("Sample complete!");
                Console.WriteLine("Press ENTER to continue");
                Console.ReadLine();
            }
            catch (AmazonServiceException e) { Console.WriteLine(e.Message); }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }

        private static void DeleteTable(string tableName)
        {
            try
            {
                var deleteTableResponse = client.DeleteTable(new DeleteTableRequest()
                {
                    TableName = tableName
                });
                WaitTillTableDeleted(client, tableName, deleteTableResponse);
            }
            catch (ResourceNotFoundException)
            {
                // There is no such table.
            }
        }

        private static void CreateTableProductCatalog()
        {
```

```
            string tableName = "ProductCatalog";

            var response = client.CreateTable(new CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                            {
                                new AttributeDefinition
                                {
                                    AttributeName = "Id",
                                    AttributeType = "N"
                                }
                            },
                KeySchema = new List<KeySchemaElement>()
                            {
                                new KeySchemaElement
                                {
                                    AttributeName = "Id",
                                    KeyType = "HASH"
                                }
                            },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 5
                }
            });

            WaitTillTableCreated(client, tableName, response);
        }

        private static void CreateTableForum()
        {
            string tableName = "Forum";

            var response = client.CreateTable(new CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                            {
                                new AttributeDefinition
                                {
                                    AttributeName = "Name",
                                    AttributeType = "S"
```

```
                            }
                        },
            KeySchema = new List<KeySchemaElement>()
                        {
                            new KeySchemaElement
                            {
                                AttributeName = "Name", // forum Title
                                KeyType = "HASH"
                            }
                        },
            ProvisionedThroughput = new ProvisionedThroughput
            {
                ReadCapacityUnits = 10,
                WriteCapacityUnits = 5
            }
        });

        WaitTillTableCreated(client, tableName, response);
    }

    private static void CreateTableThread()
    {
        string tableName = "Thread";

        var response = client.CreateTable(new CreateTableRequest
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
                        {
                            new AttributeDefinition
                            {
                                AttributeName = "ForumName", // Hash attribute
                                AttributeType = "S"
                            },
                            new AttributeDefinition
                            {
                                AttributeName = "Subject",
                                AttributeType = "S"
                            }
                        },
            KeySchema = new List<KeySchemaElement>()
                        {
                            new KeySchemaElement
                            {
```

```
                                AttributeName = "ForumName", // Hash attribute
                                KeyType = "HASH"
                            },
                            new KeySchemaElement
                            {
                                AttributeName = "Subject", // Range attribute
                                KeyType = "RANGE"
                            }
                        },
            ProvisionedThroughput = new ProvisionedThroughput
            {
                ReadCapacityUnits = 10,
                WriteCapacityUnits = 5
            }
        });

        WaitTillTableCreated(client, tableName, response);
    }

    private static void CreateTableReply()
    {
        string tableName = "Reply";
        var response = client.CreateTable(new CreateTableRequest
        {
            TableName = tableName,
            AttributeDefinitions = new List<AttributeDefinition>()
                        {
                            new AttributeDefinition
                            {
                                AttributeName = "Id",
                                AttributeType = "S"
                            },
                            new AttributeDefinition
                            {
                                AttributeName = "ReplyDateTime",
                                AttributeType = "S"
                            },
                            new AttributeDefinition
                            {
                                AttributeName = "PostedBy",
                                AttributeType = "S"
                            }
                        },
            KeySchema = new List<KeySchemaElement>()
```

```
                                {
                                    new KeySchemaElement()
                                    {
                                        AttributeName = "Id",
                                        KeyType = "HASH"
                                    },
                                    new KeySchemaElement()
                                    {
                                        AttributeName = "ReplyDateTime",
                                        KeyType = "RANGE"
                                    }
                                },
                LocalSecondaryIndexes = new List<LocalSecondaryIndex>()
                                {
                                    new LocalSecondaryIndex()
                                    {
                                        IndexName = "PostedBy_index",


                                        KeySchema = new List<KeySchemaElement>() {
                                            new KeySchemaElement() {
                                                AttributeName = "Id", KeyType = "HASH"
                                            },
                                            new KeySchemaElement() {
                                                AttributeName = "PostedBy", KeyType =
 "RANGE"
                                            }
                                        },
                                        Projection = new Projection() {
                                            ProjectionType = ProjectionType.KEYS_ONLY
                                        }
                                    }
                                },
                ProvisionedThroughput = new ProvisionedThroughput
                {
                    ReadCapacityUnits = 10,
                    WriteCapacityUnits = 5
                }
            });

            WaitTillTableCreated(client, tableName, response);
        }
```

```
        private static void WaitTillTableCreated(AmazonDynamoDBClient client, string
tableName,
                                 CreateTableResponse response)
        {
            var tableDescription = response.TableDescription;

            string status = tableDescription.TableStatus;

            Console.WriteLine(tableName + " - " + status);

            // Let us wait until table is created. Call DescribeTable.
            while (status != "ACTIVE")
            {
                System.Threading.Thread.Sleep(5000); // Wait 5 seconds.
                try
                {
                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });
                    Console.WriteLine("Table name: {0}, status: {1}",
res.Table.TableName,
                                      res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
                // Try-catch to handle potential eventual-consistency issue.
                catch (ResourceNotFoundException)
                { }
            }
        }

        private static void WaitTillTableDeleted(AmazonDynamoDBClient client, string
tableName,
                                 DeleteTableResponse response)
        {
            var tableDescription = response.TableDescription;

            string status = tableDescription.TableStatus;

            Console.WriteLine(tableName + " - " + status);

            // Let us wait until table is created. Call DescribeTable
            try
            {
```

```
                while (status == "DELETING")
                {
                    System.Threading.Thread.Sleep(5000); // wait 5 seconds

                    var res = client.DescribeTable(new DescribeTableRequest
                    {
                        TableName = tableName
                    });
                    Console.WriteLine("Table name: {0}, status: {1}",
res.Table.TableName,
                              res.Table.TableStatus);
                    status = res.Table.TableStatus;
                }
            }
            catch (ResourceNotFoundException)
            {
                // Table deleted.
            }
        }

        private static void LoadSampleProducts()
        {
            Table productCatalogTable = Table.LoadTable(client, "ProductCatalog");
            // ********** Add Books *********************
            var book1 = new Document();
            book1["Id"] = 101;
            book1["Title"] = "Book 101 Title";
            book1["ISBN"] = "111-1111111111";
            book1["Authors"] = new List<string> { "Author 1" };
            book1["Price"] = -2; // *** Intentional value. Later used to illustrate
scan.
            book1["Dimensions"] = "8.5 x 11.0 x 0.5";
            book1["PageCount"] = 500;
            book1["InPublication"] = true;
            book1["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book1);

            var book2 = new Document();

            book2["Id"] = 102;
            book2["Title"] = "Book 102 Title";
            book2["ISBN"] = "222-2222222222";
            book2["Authors"] = new List<string> { "Author 1", "Author 2" }; ;
            book2["Price"] = 20;
```

```
            book2["Dimensions"] = "8.5 x 11.0 x 0.8";
            book2["PageCount"] = 600;
            book2["InPublication"] = true;
            book2["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book2);

            var book3 = new Document();
            book3["Id"] = 103;
            book3["Title"] = "Book 103 Title";
            book3["ISBN"] = "333-3333333333";
            book3["Authors"] = new List<string> { "Author 1", "Author2", "Author
3" }; ;
            book3["Price"] = 2000;
            book3["Dimensions"] = "8.5 x 11.0 x 1.5";
            book3["PageCount"] = 700;
            book3["InPublication"] = false;
            book3["ProductCategory"] = "Book";
            productCatalogTable.PutItem(book3);

            // ************ Add bikes. ******************
            var bicycle1 = new Document();
            bicycle1["Id"] = 201;
            bicycle1["Title"] = "18-Bike 201"; // size, followed by some title.
            bicycle1["Description"] = "201 description";
            bicycle1["BicycleType"] = "Road";
            bicycle1["Brand"] = "Brand-Company A"; // Trek, Specialized.
            bicycle1["Price"] = 100;
            bicycle1["Color"] = new List<string> { "Red", "Black" };
            bicycle1["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle1);

            var bicycle2 = new Document();
            bicycle2["Id"] = 202;
            bicycle2["Title"] = "21-Bike 202Brand-Company A";
            bicycle2["Description"] = "202 description";
            bicycle2["BicycleType"] = "Road";
            bicycle2["Brand"] = "";
            bicycle2["Price"] = 200;
            bicycle2["Color"] = new List<string> { "Green", "Black" };
            bicycle2["ProductCategory"] = "Bicycle";
            productCatalogTable.PutItem(bicycle2);

            var bicycle3 = new Document();
            bicycle3["Id"] = 203;
```

```
            bicycle3["Title"] = "19-Bike 203";
            bicycle3["Description"] = "203 description";
            bicycle3["BicycleType"] = "Road";
            bicycle3["Brand"] = "Brand-Company B";
            bicycle3["Price"] = 300;
            bicycle3["Color"] = new List<string> { "Red", "Green", "Black" };
            bicycle3["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle3);

            var bicycle4 = new Document();
            bicycle4["Id"] = 204;
            bicycle4["Title"] = "18-Bike 204";
            bicycle4["Description"] = "204 description";
            bicycle4["BicycleType"] = "Mountain";
            bicycle4["Brand"] = "Brand-Company B";
            bicycle4["Price"] = 400;
            bicycle4["Color"] = new List<string> { "Red" };
            bicycle4["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle4);

            var bicycle5 = new Document();
            bicycle5["Id"] = 205;
            bicycle5["Title"] = "20-Title 205";
            bicycle4["Description"] = "205 description";
            bicycle5["BicycleType"] = "Hybrid";
            bicycle5["Brand"] = "Brand-Company C";
            bicycle5["Price"] = 500;
            bicycle5["Color"] = new List<string> { "Red", "Black" };
            bicycle5["ProductCategory"] = "Bike";
            productCatalogTable.PutItem(bicycle5);
        }

        private static void LoadSampleForums()
        {
            Table forumTable = Table.LoadTable(client, "Forum");

            var forum1 = new Document();
            forum1["Name"] = "Amazon DynamoDB"; // PK
            forum1["Category"] = "Amazon Web Services";
            forum1["Threads"] = 2;
            forum1["Messages"] = 4;
            forum1["Views"] = 1000;

            forumTable.PutItem(forum1);
```

```
            var forum2 = new Document();
            forum2["Name"] = "Amazon S3"; // PK
            forum2["Category"] = "Amazon Web Services";
            forum2["Threads"] = 1;

            forumTable.PutItem(forum2);
        }

        private static void LoadSampleThreads()
        {
            Table threadTable = Table.LoadTable(client, "Thread");

            // Thread 1.
            var thread1 = new Document();
            thread1["ForumName"] = "Amazon DynamoDB"; // Hash attribute.
            thread1["Subject"] = "DynamoDB Thread 1"; // Range attribute.
            thread1["Message"] = "DynamoDB thread 1 message text";
            thread1["LastPostedBy"] = "User A";
            thread1["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(14,
0, 0, 0));
            thread1["Views"] = 0;
            thread1["Replies"] = 0;
            thread1["Answered"] = false;
            thread1["Tags"] = new List<string> { "index", "primarykey", "table" };

            threadTable.PutItem(thread1);

            // Thread 2.
            var thread2 = new Document();
            thread2["ForumName"] = "Amazon DynamoDB"; // Hash attribute.
            thread2["Subject"] = "DynamoDB Thread 2"; // Range attribute.
            thread2["Message"] = "DynamoDB thread 2 message text";
            thread2["LastPostedBy"] = "User A";
            thread2["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(21,
0, 0, 0));
            thread2["Views"] = 0;
            thread2["Replies"] = 0;
            thread2["Answered"] = false;
            thread2["Tags"] = new List<string> { "index", "primarykey", "rangekey" };

            threadTable.PutItem(thread2);

            // Thread 3.
```

```
            var thread3 = new Document();
            thread3["ForumName"] = "Amazon S3"; // Hash attribute.
            thread3["Subject"] = "S3 Thread 1"; // Range attribute.
            thread3["Message"] = "S3 thread 3 message text";
            thread3["LastPostedBy"] = "User A";
            thread3["LastPostedDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7, 0,
0, 0));
            thread3["Views"] = 0;
            thread3["Replies"] = 0;
            thread3["Answered"] = false;
            thread3["Tags"] = new List<string> { "largeobjects", "multipart upload" };
            threadTable.PutItem(thread3);
        }

        private static void LoadSampleReplies()
        {
            Table replyTable = Table.LoadTable(client, "Reply");

            // Reply 1 - thread 1.
            var thread1Reply1 = new Document();
            thread1Reply1["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
            thread1Reply1["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(21,
0, 0, 0)); // Range attribute.
            thread1Reply1["Message"] = "DynamoDB Thread 1 Reply 1 text";
            thread1Reply1["PostedBy"] = "User A";

            replyTable.PutItem(thread1Reply1);

            // Reply 2 - thread 1.
            var thread1reply2 = new Document();
            thread1reply2["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
            thread1reply2["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(14,
0, 0, 0)); // Range attribute.
            thread1reply2["Message"] = "DynamoDB Thread 1 Reply 2 text";
            thread1reply2["PostedBy"] = "User B";

            replyTable.PutItem(thread1reply2);

            // Reply 3 - thread 1.
            var thread1Reply3 = new Document();
            thread1Reply3["Id"] = "Amazon DynamoDB#DynamoDB Thread 1"; // Hash
attribute.
```

```
            thread1Reply3["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7,
 0, 0, 0)); // Range attribute.
            thread1Reply3["Message"] = "DynamoDB Thread 1 Reply 3 text";
            thread1Reply3["PostedBy"] = "User B";

            replyTable.PutItem(thread1Reply3);

            // Reply 1 - thread 2.
            var thread2Reply1 = new Document();
            thread2Reply1["Id"] = "Amazon DynamoDB#DynamoDB Thread 2"; // Hash
 attribute.
            thread2Reply1["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(7,
 0, 0, 0)); // Range attribute.
            thread2Reply1["Message"] = "DynamoDB Thread 2 Reply 1 text";
            thread2Reply1["PostedBy"] = "User A";



            replyTable.PutItem(thread2Reply1);

            // Reply 2 - thread 2.
            var thread2Reply2 = new Document();
            thread2Reply2["Id"] = "Amazon DynamoDB#DynamoDB Thread 2"; // Hash
 attribute.
            thread2Reply2["ReplyDateTime"] = DateTime.UtcNow.Subtract(new TimeSpan(1,
 0, 0, 0)); // Range attribute.
            thread2Reply2["Message"] = "DynamoDB Thread 2 Reply 2 text";
            thread2Reply2["PostedBy"] = "User A";

            replyTable.PutItem(thread2Reply2);
        }
    }
}
```

Aplicación de ejemplo de DynamoDB con AWS SDK for Python (Boto): Tic-Tac-Toe (Tres en raya)

El juego Tic-Tac-Toe (Tres en raya) es un ejemplo de aplicación web creada en Amazon DynamoDB. La aplicación utiliza AWS SDK for Python (Boto) para llevar a cabo las llamadas a DynamoDB necesarias para almacenar los datos del juego en una tabla de DynamoDB, y el marco web de Python, Flask, para mostrar el desarrollo íntegro de la aplicación en DynamoDB, lo que incluye cómo se modelan los datos. Además, en el ejemplo se muestran las prácticas recomendadas en lo que

respecta al modelado de datos en DynamoDB; incluyendo la tabla que se crea para la aplicación del juego, la clave principal que se define, los índices adicionales que se necesitan según los requisitos de las consultas y el uso de atributos de valores concatenados.

A continuación se indica cómo se juega a la aplicación Tic-Tac-Toe en la Web:

1. Inicie sesión en la página de inicio de la aplicación.
2. A continuación, invite a otro usuario a jugar una partida como su contrincante.

   Hasta que otro usuario acepta la invitación, el estado de la partida es `PENDING`. Una vez que un contrincante ha aceptado la invitación, el estado de la partida cambia a `IN_PROGRESS`.
3. La partida comienza una vez que su contrincante ha iniciado sesión y aceptado la invitación.
4. La aplicación almacena todas las jugadas de las partidas y la información de estado en una tabla de DynamoDB.
5. La partida termina cuando un jugador gana o si se produce un empate, en cuyo caso se establece el estado de la partida en `FINISHED`.

El ejercicio completa de creación de la aplicación se describe en varios pasos:

- Paso 1: implementar y probar localmente – en esta sección, usted descarga, implementa y prueba la aplicación en su ordenador local. Creará las tablas requeridas en la versión descargable de DynamoDB.
- Paso 2: examinar el modelo de datos y los detalles de implementación – en esta sección se describe primero con detalle el modelo de datos, incluidos los índices y el uso del atributo de valores concatenados. A continuación, se explica el funcionamiento de la aplicación.
- Paso 3: implementar en producción mediante el servicio de DynamoDB – esta sección se centra en las consideraciones de implementación en un entorno de producción. En este paso, usted crea una tabla mediante el servicio de Amazon DynamoDB e implementa la aplicación con AWS Elastic Beanstalk. Cuando la aplicación se encuentra en producción, se conceden además los permisos adecuados para que la aplicación pueda acceder a la tabla de DynamoDB. Las instrucciones de esta sección le guiarán a lo largo de todo el proceso de implementación en producción.
- Paso 4: limpie los recursos – esta sección resalta las áreas que no se abordan en este ejemplo. Además, la sección proporciona los pasos que deben seguirse para eliminar los recursos de AWS que se han creado en los pasos anteriores, con el fin de evitar la incursión en algún cargo.

Paso 1: implementar y probar localmente

Temas

- 1.1: descargar e instalar los paquetes obligatorios
- 1.2: probar la aplicación del juego

En este paso se descarga, implementa y prueba la aplicación del juego de tres en raya en el equipo local. En lugar de utilizar el servicio web de Amazon DynamoDB, descargue DynamoDB en su ordenador y cree en él la tabla requerida.

1.1: descargar e instalar los paquetes obligatorios

Necesitará lo siguiente para probar localmente esta aplicación:

- Python
- Flask (un micromarco de trabajo para Python)
- AWS SDK for Python (Boto)
- Ejecución de DynamoDB en su ordenador
- Git

Para obtener estas herramientas, haga lo siguiente:

1. Instalación de Python. Para ver instrucciones paso a paso, consulte Download Python.

   La aplicación Tic-Tac-Toe se ha probado con Python versión 2.7.
2. Utilice Python Package Installer (PIP) para instalar Flask y el AWS SDK for Python (Boto):
   - Instale PIP.

     Para ver instrucciones, consulte Install PIP. En la página de instalación, seleccione el enlace get-pip.py y, a continuación, guarde el archivo. Después abra un terminal de comandos como administrador y escriba lo siguiente en el símbolo del sistema.

     ```
     python.exe get-pip.py
     ```

     En Linux, no se especifica la extensión `.exe`. Solo se especifica `python get-pip.py`.
   - Utilice PIP para instalar los paquetes Flask y Boto mediante el siguiente código.

```
pip install Flask
pip install boto
pip install configparser
```

3. Descargue DynamoDB en su ordenador. Para obtener instrucciones sobre cómo ejecutarlo, consulte Configuración de la versión de DynamoDB local (versión descargable) .
4. Descargue la aplicación Tic-Tac-Toe:
   a. Instala Git Para ver instrucciones, consulte git downloads.
   b. Para descargar la aplicación, ejecute el siguiente código:

```
git clone https://github.com/awslabs/dynamodb-tictactoe-example-app.git
```

1.2: probar la aplicación del juego

Para probar la aplicación Tic-Tac-Toe (Tres en raya), debe ejecutar DynamoDB localmente en su ordenador.

Para ejecutar la aplicación tic-tac-toe (tres en raya)

1. Inicie DynamoDB.
2. Inicie el servidor web de la aplicación Tic-Tac-Toe.

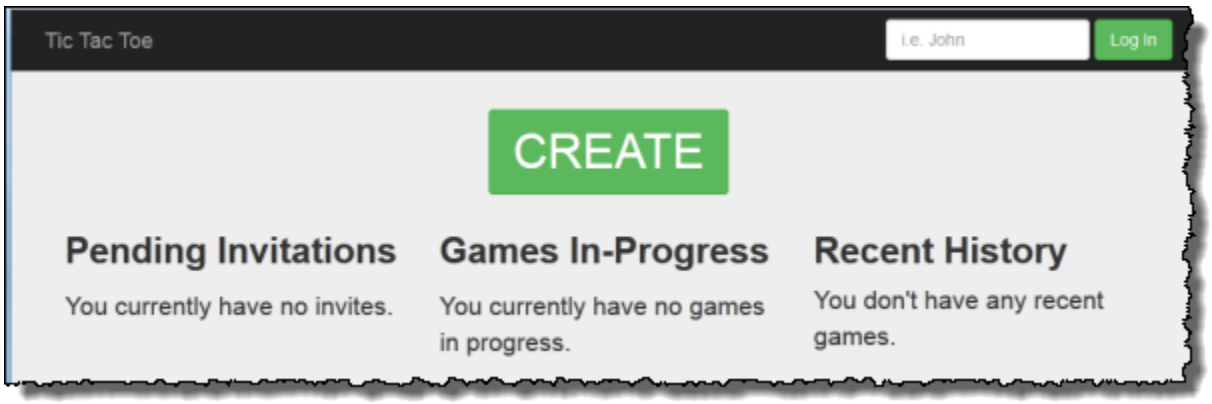
   Para ello, abra un terminal de comandos, diríjase a la carpeta en la que ha descargado la aplicación Tic-Tac-Toe (Tres en raya) y ejecute la aplicación localmente con el código siguiente.

```
python.exe application.py --mode local --serverPort 5000 --port 8000
```

   En Linux, no se especifica la extensión `.exe`.
3. Abra el navegador web y escriba lo siguiente.

```
http://localhost:5000/
```

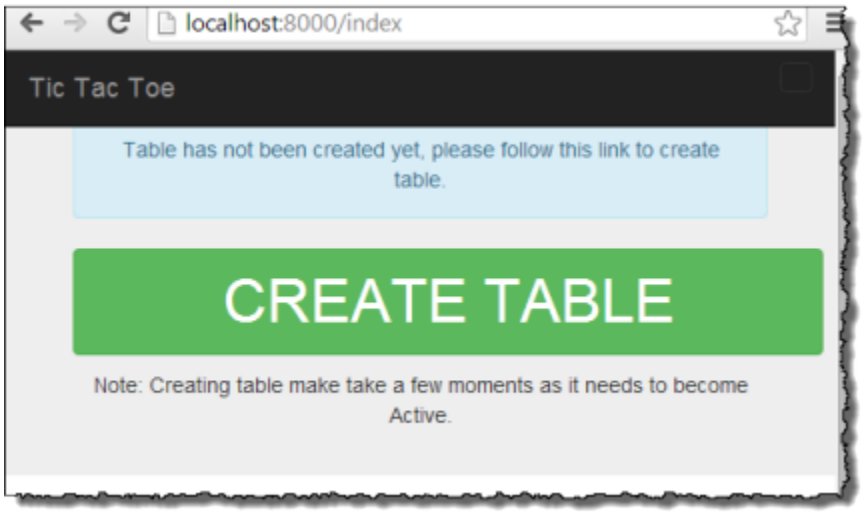
   En el navegador aparece la página de inicio.



4. Escriba **`user1`** en el cuadro Log in (Iniciar sesión) para iniciar sesión como user1.

> **Note**
>
> En este ejemplo de aplicación no se lleva a cabo la autenticación del usuario. El identificador de usuario se utiliza solamente para identificar a los jugadores. Si dos jugadores inician sesión con el mismo alias, la aplicación funciona como si estuviesen jugando en dos navegadores distintos.

5. Si es la primera vez que inicia el juego, aparece una página que le pide que cree la tabla requerida (`Games`) en DynamoDB. Seleccione CREATE TABLE (CREAR TABLA).



6. Elija CREATE (CREAR) para crear la primera partida de tres en raya.
7. Escriba **`user2`** en el cuadro Choose an Opponent (Elegir un contrincante) y elija Create Game! (Crear partida)

Se crea la partida agregando un elemento en la tabla `Games`. El estado de la partida se establece en `PENDING`.

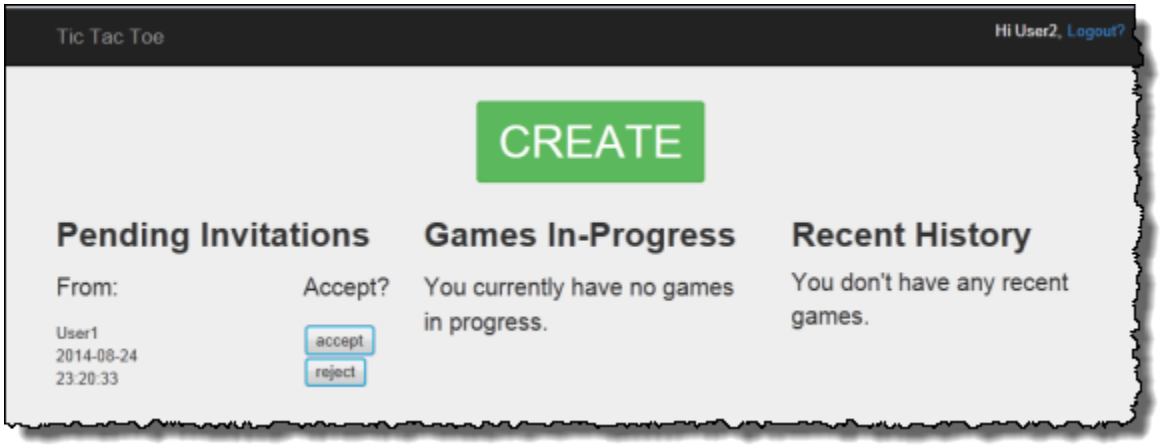
8. Abra otra ventana del navegador y escriba lo siguiente.

```
http://localhost:5000/
```

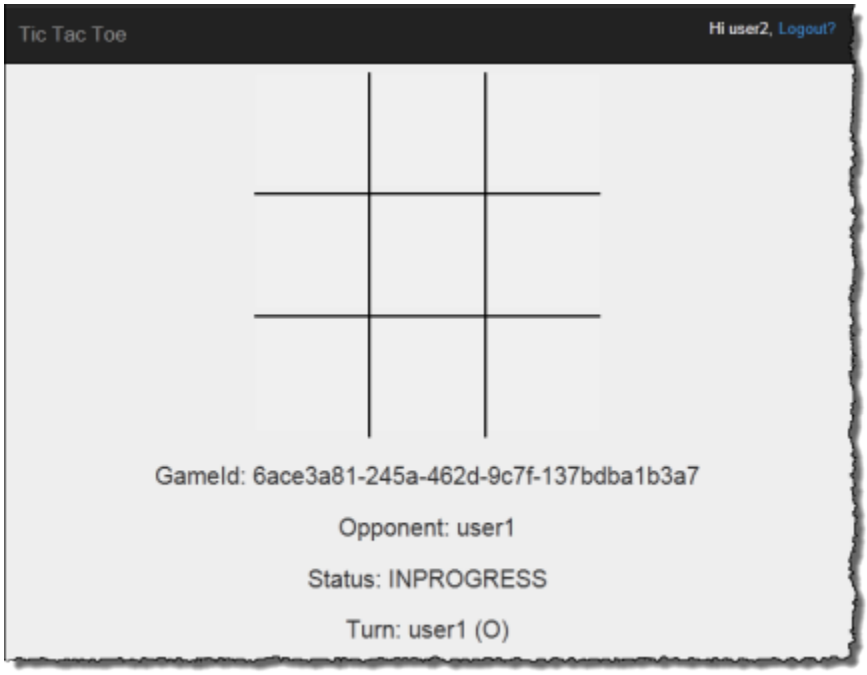
El navegador transmite información a través de las cookies, por lo que ha de utilizar el modo de incógnito o la navegación privada para que las cookies no se conserven.

9. Inicie sesión como user2.

Aparecerá una página con una invitación pendiente de user1.



10. Elija accept (aceptar) para aceptar la invitación.



Aparecerá la página de la partida con una cuadrícula de tres en raya vacía. En la página se muestra también información pertinente sobre la partida, como el Id. de la partida, a quién le toca jugar y el estado de la partida.

11. Juegue la partida.

Cada vez que un usuario mueve la ficha, el servicio web envía una solicitud a DynamoDB para actualizar de forma condicional el elemento de la partida en la tabla `Games`. Por ejemplo, las condiciones garantizan que el movimiento haya sido válido, que la casilla elegida por el usuario esté disponible o que le tocaba jugar al usuario que movió la ficha. Para cada jugada válida, la operación de actualización agrega un nuevo atributo correspondiente a la casilla seleccionada en el tablero. La operación de actualización también establece el valor del atributo existente en el usuario que puede llevar a cabo la siguiente jugada.

En la página del juego, la aplicación realiza llamadas asíncronas a JavaScript cada segundo durante un máximo de 5 minutos, para comprobar si el estado del juego en DynamoDB ha cambiado. En caso afirmativo, la aplicación actualiza la página con la nueva información. Después de 5 minutos, la aplicación deja de realizar las solicitudes y tendrá que actualizar la página si desea obtener información actualizada.

Paso 2: examinar el modelo de datos y los detalles de implementación

Temas

- 2.1: modelo de datos básicos
- 2.2: aplicación en acción (guía del código)

2.1: modelo de datos básicos

En este ejemplo de aplicación se resaltan los siguientes conceptos del modelo de datos de DynamoDB:

- Tabla: en DynamoDB, una tabla es una colección de elementos (es decir, de registros) y cada elemento es una colección de pares de nombre-valor denominados atributos.

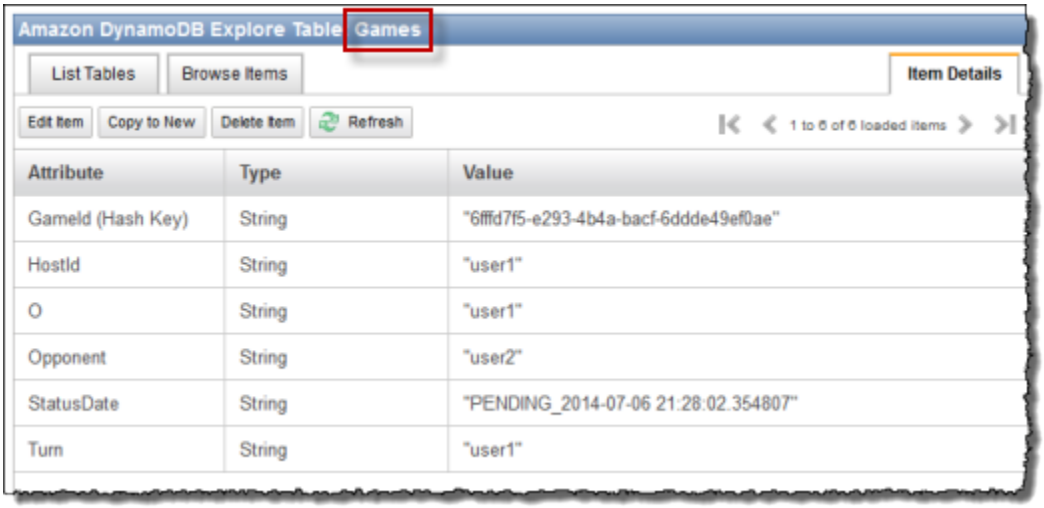
  En este ejemplo del juego de tres en raya, la aplicación almacena todos los datos de las partidas en una tabla, `Games`. La aplicación crea un elemento en la tabla por cada partida y almacena todos los datos de las partidas como atributos. Una partida de tres en raya puede incluir hasta nueve jugadas o movimientos. Dado que las tablas de DynamoDB no tienen un esquema cuando el único atributo obligatorio es la clave principal, la aplicación puede almacenar un número variable de atributos por cada elemento del juego.

  La tabla `Games` cuenta con una clave principal sencilla que consta de un solo atributo, `GameId`, de tipo String. La aplicación asigna identificador exclusivo a cada partida. Para obtener más información sobre claves principales en DynamoDB, consulte Clave principal.

  Cuando un usuario inicia una partida de tres en raya invitando a otro usuario a jugar, la aplicación crea un nuevo elemento en la tabla `Games` con atributos que almacenan los metadatos de la partida, tales como los siguientes:

  - `HostId`, usuario que inició la partida.
  - `Opponent`, usuario al que se invitó a jugar.
  - Usuario al que le toca jugar. El usuario que inicia la partida juega primero.
  - El usuario que utiliza el símbolo O en el tablero. El usuario que inicia las partidas utiliza el símbolo O.

  Además, la aplicación crea un atributo `StatusDate` concatenado que marca el estado inicial de la partida como `PENDING`. En la siguiente captura de pantalla se muestra un ejemplo de elemento tal y como aparece en la consola de DynamoDB:

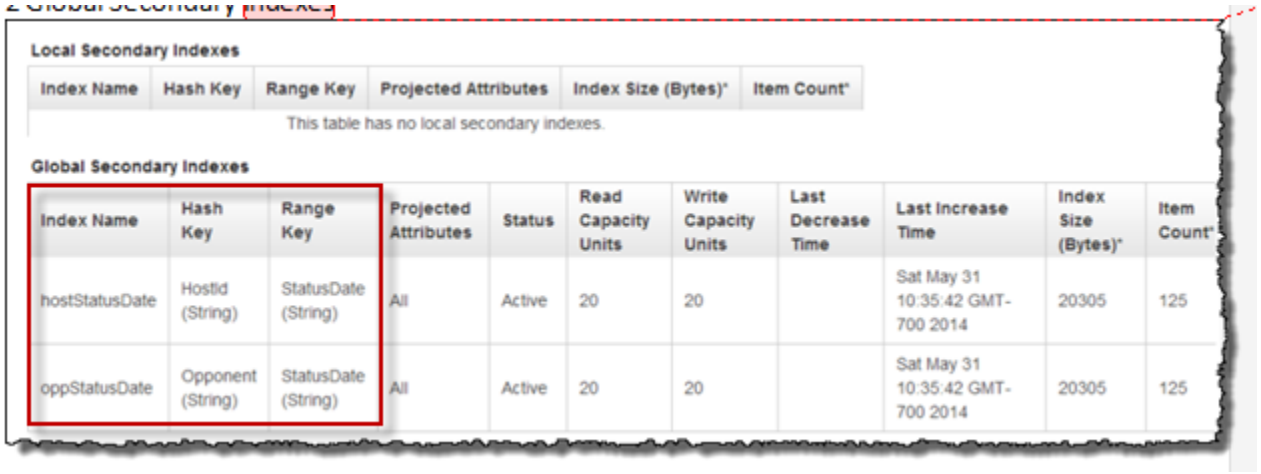


A medida que la partida avanza, la aplicación agrega un atributo a la tabla por cada jugada. El nombre del atributo es la posición en el tablero; por ejemplo, `TopLeft` o `BottomRight`. Por ejemplo, una jugada puede tener el atributo `TopLeft` con el valor `0`, un atributo `TopRight` con el valor `0` y un atributo `BottomRight` con el valor `X`. El valor del atributo puede ser `0` o `X`, según cuál sea el usuario que ha realizado la jugada. Por ejemplo, fíjese en el tablero siguiente.

- Atributos de valores concatenados: el atributo `StatusDate` ilustra un valor de atributo concatenado. En este enfoque, en lugar de crear atributos separados para almacenar el estado de la partida (`PENDING`, `IN_PROGRESS` y `FINISHED`) y la fecha (cuándo se realizó la última jugada), se combinan como un solo atributo; por ejemplo `IN_PROGRESS_2014-04-30 10:20:32`.

  A continuación, la aplicación utiliza el atributo `StatusDate` para crear índices secundarios especificando `StatusDate` como clave de ordenación del índice. El beneficio de utilizar el atributo de valores concatenados `StatusDate` se ilustra mejor en la explicación sobre índices que encontrará más adelante.

- Índices secundarios globales: puede utilizar la clave principal de la tabla, `GameId`, para consultar esa tabla de manera eficiente con el fin de encontrar un elemento del juego. Para consultar la tabla para hallar otros atributos distintos de los de clave principal, DynamoDB admite la creación de índices secundarios. En este ejemplo de aplicación, se crean los dos índices secundarios siguientes:



Local Secondary Indexes

Index Name	Hash Key	Range Key	Projected Attributes	Index Size (Bytes)*	Item Count*

This table has no local secondary indexes.

Global Secondary Indexes

Index Name	Hash Key	Range Key	Projected Attributes	Status	Read Capacity Units	Write Capacity Units	Last Decrease Time	Last Increase Time	Index Size (Bytes)*	Item Count*
hostStatusDate	HostId (String)	StatusDate (String)	All	Active	20	20		Sat May 31 10:35:42 GMT-700 2014	20305	125
oppStatusDate	Opponent (String)	StatusDate (String)	All	Active	20	20		Sat May 31 10:35:42 GMT-700 2014	20305	125

  - HostId-StatusDate-index. Este índice tiene `HostId` como clave de partición y `StatusDate` como clave de ordenación. Puede utilizar este índice para realizar una consulta sobre `HostId`, por ejemplo, para encontrar las partidas organizadas por un usuario concreto.
  - OpponentId-StatusDate-index. Este índice tiene `OpponentId` como clave de partición y `StatusDate` como clave de ordenación. Puede utilizar este índice para realizar una consulta sobre `Opponent`, por ejemplo, para encontrar las partidas en las que un usuario determinado ha sido el contrincante.

Estos índices se denominan índices secundarios globales porque su clave de partición no es la misma (`GameId`) que se usó en la clave principal de la tabla.

Tenga en cuenta que en los dos índices se especifica `StatusDate` como clave de ordenación. Al hacerlo, se habilita lo siguiente:

- Puede realizar consultas utilizando el operador de comparación `BEGINS_WITH`. Por ejemplo, puede buscar todas las partidas que tienen el atributo `IN_PROGRESS` y que ha organizado un usuario determinado. En este caso, el operador `BEGINS_WITH` comprueba los valores de `StatusDate` que comienzan por `IN_PROGRESS`.
- DynamoDB almacena los elementos en el índice de forma secuencial, según el valor de la clave de ordenación. Así pues, si todos los prefijos de estado son iguales (por ejemplo, `IN_PROGRESS`), el formato ISO utilizado para la parte de la fecha hará que los elementos se

ordenen por orden de antigüedad descendente. Este enfoque permite realizar con eficacia algunas consultas, tales como las siguientes:

- Recuperar como máximo las diez partidas `IN_PROGRESS` más recientes organizadas por el usuario que ha iniciado sesión. Para esta consulta, se especifica el índice `HostId-StatusDate-index`.
- Recuperar como máximo las diez partidas `IN_PROGRESS` más recientes en las que el usuario que ha iniciado sesión sea el contrincante. Para esta consulta, se especifica el índice `OpponentId-StatusDate-index`.

Para obtener más información acerca de los índices secundarios, consulte Mejora del acceso con índices secundarios en DynamoDB.

2.2: aplicación en acción (guía del código)

Esta aplicación tiene dos páginas principales:

- Página de inicio: en esta página se proporciona al usuario un inicio de sesión sencillo, un botón CREATE para crear una nueva partida de tres en raya, una lista de partidas en curso, el historial de partidas y todas las invitaciones pendientes para jugar.

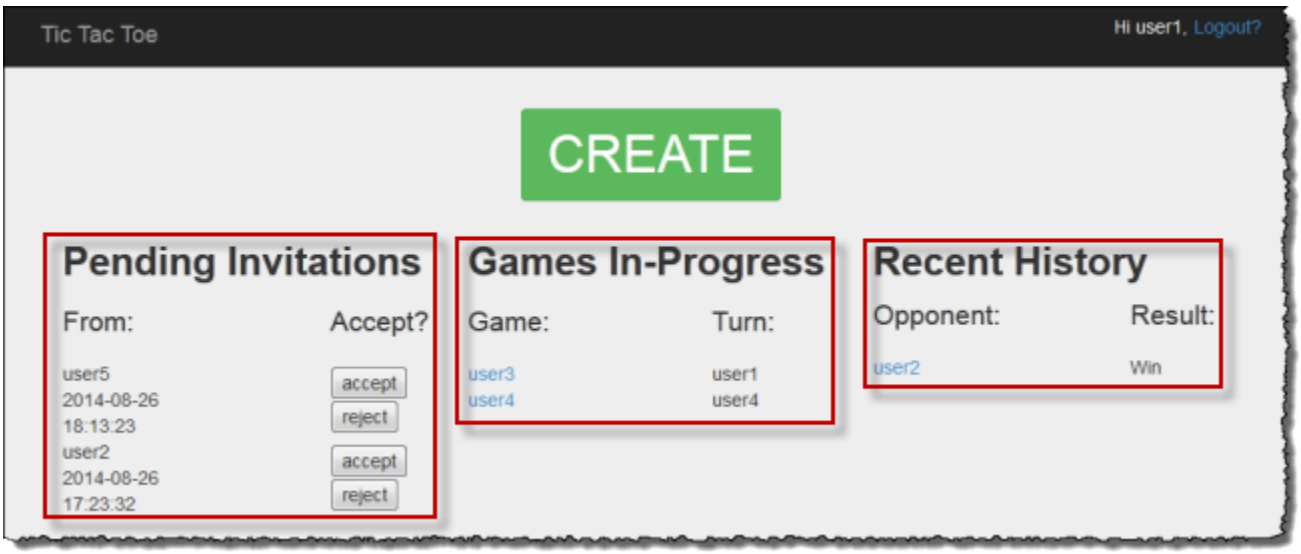
  La página de inicio no se actualiza automáticamente; debe actualizarla para renovar la información de las listas.
- Página de la partida: muestra la cuadrícula de tres en raya en la que juegan los usuarios.

  La aplicación actualiza la página de la partida automáticamente cada segundo. El código JavaScript del navegador llama al servidor web Python cada segundo para consultar la tabla Games y saber si los elementos de partidas contenidos en la tabla han cambiado. En caso afirmativo, JavaScript activa una actualización de la página para que el usuario vea el tablero con la información más reciente.

Vamos a estudiar en detalle el funcionamiento de la aplicación.

Página de inicio

Cuando el usuario inicia sesión, la aplicación muestra las tres listas de información siguientes.



- Invitaciones: esta lista muestra como máximo las diez invitaciones más recientes de otros usuarios que el usuario que ha iniciado sesión todavía no ha aceptado. En la captura de pantalla anterior, el usuario user1 tiene invitaciones pendientes de los usuarios user2 y user5.
- Juegos en progreso: esta lista muestra como máximo las 10 partidas más recientes que están en curso. Se trata de partidas en las que el usuario está jugando activamente y cuyo estado es `IN_PROGRESS`. En la captura de pantalla, el usuario user1 está jugando activamente una partida de tres en raya contra los usuarios user3 y user4.
- Historia reciente: esta lista muestra como máximo las 10 partidas más recientes que el usuario ha terminado, cuyo estado es `FINISHED`. En el juego que aparece en la captura de pantalla, el usuario user1 ha jugado anteriormente contra el usuario user2. En la lista se muestra el resultado de cada partida completada.

En el código, la función `index` (en `application.py`) realiza las tres llamadas siguientes para recuperar la información de estado de las partidas:

```
inviteGames     = controller.getGameInvites(session["username"])
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

Cada una de estas llamadas devuelve una lista de elementos de DynamoDB que están integrados en los objetos `Game`. Resulta fácil extraer datos de estos objetos en la vista. La función de índice transmite estas listas de objetos a la vista para representar el código HTML.

```
return render_template("index.html",
```

```
                    user=session["username"],
                    invites=inviteGames,
                    inprogress=inProgressGames,
                    finished=finishedGames)
```

La aplicación del juego de Tic-Tac-Toe (Tres en raya) define la clase `Game` principalmente para almacenar los datos de los juegos recuperados de DynamoDB. Estas funciones devuelven listas de objetos `Game` que le permiten aislar el resto de la aplicación del código relacionado con los elementos de Amazon DynamoDB. Por lo tanto, estas funciones le ayudan a desacoplar el código de aplicación de los detalles de la capa de almacenamiento de datos.

El patrón de arquitectura descrito aquí también se denomina patrón de interfaz de usuario (IU) de tipo modelo-vista-controlador (MVC). En este caso, las instancias de objetos `Game` (que representan datos) son el modelo y la página HTML es la vista. El controlador se divide en dos archivos. El archivo `application.py` contiene la lógica del controlador correspondiente al marco de trabajo Flask, mientras que la lógica empresarial se aísla en el archivo `gameController.py`. Es decir, la aplicación guarda todo lo que se refiere al SDK de DynamoDB en su propio archivo independiente dentro de la carpeta `dynamodb`.

Vamos a revisar las tres funciones para entender cómo consultan la tabla Games y utilizan índices secundarios globales para recuperar los datos pertinentes.

Uso de getGameInvites para obtener la lista de invitaciones de juego pendientes

La función `getGameInvites` recupera la lista de las diez invitaciones pendientes más recientes. Hay usuarios que han creado estas partidas, pero los contrincantes no han aceptado las invitaciones para jugar. En estas partidas, el estado sigue siendo `PENDING` hasta que el contrincante acepta la invitación. Si el contrincante rechaza la invitación, la aplicación eliminará el elemento correspondiente de la tabla.

La función especifica la consulta de la siguiente manera:

- Especifica el índice `OpponentId-StatusDate-index` que se debe usar con los siguientes valores de clave de índice y operadores de comparación:
  - La clave de partición es `OpponentId` y acepta la clave de índice *`user ID`*.
  - La clave de ordenación es `StatusDate` y acepta el operador de comparación y el valor de clave de índice `beginswith="PENDING_"`.

El índice `OpponentId-StatusDate-index` se utiliza para recuperar partidas a las que se ha invitado a jugar al usuario que ha iniciado sesión; es decir, en las que el usuario que ha iniciado sesión es el contrincante.

- La consulta limita el resultado a diez elementos.

```
gameInvitesIndex = self.cm.getGamesTable().query(
                                            Opponent__eq=user,
                                            StatusDate__beginswith="PENDING_",
                                            index="OpponentId-StatusDate-index",
                                            limit=10)
```

En el índice, por cada `OpponentId` (clave de partición) DynamoDB ordena los elementos según `StatusDate` (clave de ordenación). Por lo tanto, las partidas que devuelve la consulta son las diez más recientes.

Uso de getGamesWithStatus para obtener la lista de juegos con un estado determinado

Una vez que un contrincante ha aceptado una invitación a jugar, el estado de la partida cambia a `IN_PROGRESS`. Cuando la partida finaliza, el estado cambia a `FINISHED`.

Las consultas para buscar las partidas que se encuentran en curso o que ya han finalizado son iguales salvo por el valor de estado, que varía. Por consiguiente, la aplicación define la función `getGamesWithStatus`, que acepta el valor de estado como parámetro.

```
inProgressGames = controller.getGamesWithStatus(session["username"], "IN_PROGRESS")
finishedGames   = controller.getGamesWithStatus(session["username"], "FINISHED")
```

En la sección siguiente se explican las partidas en curso, pero la misma descripción es aplicable también a las partidas finalizadas.

Una lista de partidas en curso de un usuario determinado incluye los dos tipos siguientes:

- Las partidas en curso organizadas por el usuario
- Las partidas en curso en las que el usuario es el contrincante

La función `getGamesWithStatus` ejecuta las dos consultas siguientes, utilizando en cada caso el índice secundario apropiado.

- La función consulta la tabla `Games` mediante el índice `HostId-StatusDate-index`. Para el índice, la consulta especifica los valores de clave principal; es decir, los valores tanto de la clave de partición (`HostId`) como de la clave de ordenación (`StatusDate`), así como los operadores de comparación.

```
hostGamesInProgress = self.cm.getGamesTable ().query(HostId__eq=user,
                                                     StatusDate__beginswith=status,
                                                     index="HostId-StatusDate-index",
                                                     limit=10)
```

  Fíjese en la sintaxis de Python para los operadores de comparación:

  - `HostId__eq=user` especifica el operador de comparación de igualdad.
  - `StatusDate__beginswith=status` especifica el operador de comparación `BEGINS_WITH`.

- La función consulta la tabla `Games` mediante el índice `OpponentId-StatusDate-index`.

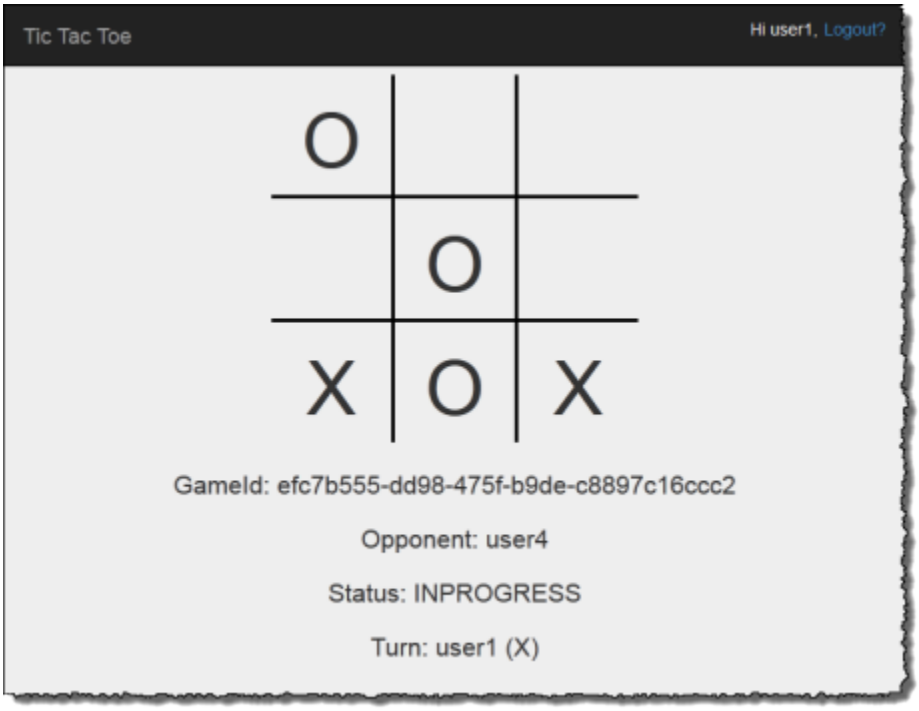
```
oppGamesInProgress = self.cm.getGamesTable().query(Opponent__eq=user,
                                                   StatusDate__beginswith=status,
                                                   index="OpponentId-StatusDate-index",
                                                   limit=10)
```

- A continuación, la función combina ambas listas, las ordena, crea una lista de entre 0 y 10 elementos que contiene los primeros objetos `Game` y se la devuelve a la función de llamada (es decir, al índice).

```
games = self.mergeQueries(hostGamesInProgress,
                          oppGamesInProgress)
return games
```

Página del juego

La página de la partida es donde el usuario juega las partidas de tres en raya. Muestra la cuadrícula del juego, además de la información pertinente sobre el juego. En la siguiente captura de pantalla se muestra un ejemplo de partida en curso:



La aplicación muestra la página de la partida en las siguientes situaciones:

- Cuando el usuario ha creado una partida e invitado a otro usuario a jugar.

  En este caso, la página muestra al usuario como anfitrión y el estado de la partida `PENDING`, mientras espera a que el contrincante acepte.
- Cuando el usuario ha aceptado una de las invitaciones pendientes de la página de inicio.

  En este caso, la página muestra al usuario como contrincante y el estado de la partida `IN_PROGRESS`.

Cuando el usuario selecciona una opción en el tablero, se genera una solicitud `POST` de formulario a la aplicación. Es decir, Flask llama a la función `selectSquare` (en `application.py`) con los datos del formulario HTML. Esta función, a su vez, llama a la función `updateBoardAndTurn` (en `gameController.py`) para actualizar el elemento de partida como se indica a continuación:

- Agrega un nuevo atributo específico de la jugada.
- Actualiza el valor del atributo `Turn` con el valor del usuario al que le toca jugar a continuación.

```
controller.updateBoardAndTurn(item, value, session["username"])
```

La función devuelve true si el elemento se actualizó correctamente; en caso contrario, devuelve false. Tenga en cuenta lo siguiente en relación con la función `updateBoardAndTurn`:

- La función llama a la función `update_item` del SDK para Python con el fin de realizar un conjunto de actualizaciones finitas de un elemento existente. La función se mapea a la operación `UpdateItem` en DynamoDB. Para obtener más información, consulte UpdateItem.

> **Note**
>
> La diferencia entre las operaciones `UpdateItem` y `PutItem` es que `PutItem` sustituye al elemento completo. Para obtener más información, consulte PutItem.

Para la llamada a `update_item`, el código identifica lo siguiente:

- Clave principal de la tabla `Games` (es decir, `ItemId`).

```
key = { "GameId" : { "S" : gameId } }
```

- Nuevo atributo que se va a agregar, específico de la jugada del usuario actual, y su valor (por ejemplo, `TopLeft="X"`).

```
attributeUpdates = {
    position : {
        "Action" : "PUT",
        "Value" : { "S" : representation }
    }
}
```

- Condiciones que deben cumplirse para que la actualización se lleve a cabo:
  - La partida debe estar en curso. Es decir, el valor del atributo `StatusDate` debe comenzar por `IN_PROGRESS`.
  - El turno actual debe ser válido para el usuario según lo especificado por el atributo `Turn`.
  - La casilla elegida por el usuario debe estar disponible. Es decir, el atributo correspondiente a la casilla no debe existir.

```
expectations = {"StatusDate" : {"AttributeValueList": [{"S" : "IN_PROGRESS_"}],
    "ComparisonOperator": "BEGINS_WITH"},
    "Turn" : {"Value" : {"S" : current_player}},
```

```
    position : {"Exists" : False}}
```

Ahora, la función llama a `update_item` para actualizar el elemento.

```
self.cm.db.update_item("Games", key=key,
    attribute_updates=attributeUpdates,
    expected=expectations)
```

Una vez que la función devuelva el resultado, la función `selectSquare` llama a "redirect", tal y como se indica en el ejemplo siguiente.

```
redirect("/game="+gameId)
```

Esta llamada hace que el navegador se actualice. Durante esta actualización, la aplicación comprueba si la partida ha terminado con un ganador o en empate. En caso afirmativo, la aplicación actualiza el elemento de partida en consecuencia.

Paso 3: implementar en producción mediante el servicio de DynamoDB

Temas

- 3.1: crear un rol de IAM para Amazon EC2
- 3.2: crear la tabla de juegos en Amazon DynamoDB
- 3.3: agrupar e implementar el código de la aplicación tic-tac-toe (tres en raya)
- 3.4: configurar el entorno de AWS Elastic Beanstalk

En las secciones anteriores, hemos implementado y probado la aplicación Tic-Tac-Toe (Tres en raya) localmente en su equipo utilizando DynamoDB Local. Ahora, vamos a implementarla en un entorno de producción, del siguiente modo:

- Implemente la aplicación con AWS Elastic Beanstalk, un servicio fácil de utilizar para implementar y ampliar servicios y aplicaciones web. Para obtener más información, consulte Implementar una aplicación flask en AWS Elastic Beanstalk.

  Elastic Beanstalk lanzará una o varias instancias Amazon Elastic Compute Cloud (Amazon EC2), que configura a través de Elastic Beanstalk, en las que se ejecutará su aplicación Tic-Tac-Toe (Tres en raya).

- Con el servicio de Amazon DynamoDB, cree una tabla `Games` que exista en AWS, en lugar de existir localmente en su ordenador.

Además, tendrá que configurar los permisos. Todos los recursos de AWS que cree, tales como la tabla `Games` en DynamoDB, son privados de forma predeterminada. Solo el propietario del recurso, es decir, la cuenta de AWS que ha creado la tabla `Games`, puede obtener acceso a esta tabla. Por lo tanto, la aplicación Tic-Tac-Toe (Tres en raya) no puede actualizar la tabla `Games` de forma predeterminada.

Para conceder los permisos necesarios, cree un rol de AWS Identity and Access Management (IAM) y concédale permisos para tener acceso a la tabla `Games`. Primero, la instancia de Amazon EC2 asume este rol. En respuesta, AWS devuelve credenciales de seguridad temporales que la instancia de Amazon EC2 puede utilizar para actualizar la tabla `Games` en nombre de la aplicación Tic-Tac-Toe (Tres en raya). Al configurar su aplicación en Elastic Beanstalk, debe especificar el rol de IAM que la instancia o instancias de Amazon EC2 pueden asumir. Para obtener más información acerca de los roles de IAM, consulte Roles de IAM para Amazon EC2 en la Guía del usuario de Amazon EC2.

> **Note**
>
> Antes de crear instancias de Amazon EC2 para la aplicación Tic-Tac-Toe (Tres en raya), primero debe decidir en qué región de AWS desea que Elastic Beanstalk las cree. Después de crear la aplicación de Elastic Beanstalk, proporcione el mismo nombre de región y el mismo punto de enlace en un archivo de configuración. La aplicación Tic-Tac-Toe (Tres en raya) utiliza la información de este archivo para crear la tabla `Games` y enviar las solicitudes subsiguientes en una región de AWS específica. Tanto la tabla `Games` de DynamoDB como las instancias de Amazon EC2 que Elastic Beanstalk lanza deben estar en la misma región. Para ver una lista de regiones disponibles, consulte Amazon DynamoDB en la Referencia general de Amazon Web Services.

Resumiendo, debe hacer lo siguiente para implementar la aplicación Tic-Tac-Toe en un entorno de producción:

1. Cree un rol de IAM mediante el servicio IAM. Debe adjuntar una política a este rol que conceda permisos para realizar las acciones de DynamoDB que se requieren para obtener acceso a la tabla `Games`.

2. Empaquete el código de la aplicación Tic-Tac-Toe y un archivo de configuración y cree un archivo `.zip`. Utilice este archivo `.zip` para proporcionar el código de la aplicación Tic-Tac-Toe (Tres en raya) a Elastic Beanstalk y ponerlo en sus servidores. Para obtener más información sobre cómo crear una agrupación, consulte Creación de una agrupación de origen de aplicación en la Guía para desarrolladores de AWS Elastic Beanstalk.

   En el archivo de configuración (`beanstalk.config`), se indica la información sobre la región y el punto de enlace de AWS. La aplicación Tic-Tac-Toe (Tres en raya) utiliza esta información para determinar con qué región de DynamoDB debe comunicarse.
3. Configure el entorno de Elastic Beanstalk. Elastic Beanstalk lanzará una o varias instancias de Amazon EC2 e implementará el paquete de la aplicación Tic-Tac-Toe (Tres en raya) en ellas. Una vez que el entorno de Elastic Beanstalk esté preparado, deberá indicar el nombre del archivo de configuración agregando la variable de entorno `CONFIG_FILE`.
4. Crear una tabla de DynamoDB Con el servicio de Amazon DynamoDB, cree una tabla `Games` en AWS, en lugar de localmente en su ordenador. Recuerde que esta tabla posee una clave principal simple que consta de la clave de partición `GameId`, de tipo String.
5. Pruebe el juego en el entorno de producción.

3.1: crear un rol de IAM para Amazon EC2

La creación de un rol de IAM del tipo Amazon EC2 permitirá que la instancia de Amazon EC2 en la que se ejecuta la aplicación Tic-Tac-Toe (Tres en raya) asuma el rol correcto y realice solicitudes a la aplicación para obtener acceso a la tabla `Games`. Al crear el rol, elija la opción Custom Policy (Política personalizada) y copie y pegue la siguiente política.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Action":[
            "dynamodb:ListTables"
         ],
         "Effect":"Allow",
         "Resource":"*"
      },
      {
```

```
            "Action":[
               "dynamodb:*"
            ],
            "Effect":"Allow",
            "Resource":[
               "arn:aws:dynamodb:us-west-2:922852403271:table/Games",
               "arn:aws:dynamodb:us-west-2:922852403271:table/Games/index/*"
            ]
         }
      ]
}
```

Para obtener más instrucciones, consulte Creación de un rol para un servicio de AWS (Consola de administración de AWS) en la Guía del usuario de IAM.

3.2: crear la tabla de juegos en Amazon DynamoDB

La tabla `Games` en DynamoDB almacena datos del juego. Si la tabla no existiera, la aplicación la crearía automáticamente. En este caso, permita que la aplicación cree la tabla `Games`.

3.3: agrupar e implementar el código de la aplicación tic-tac-toe (tres en raya)

Si ha seguido los pasos de este ejemplo, ya tendrá la aplicación Tic-Tac-Toe descargada. Si no es así, descargue la aplicación y extraiga todos los archivos en una carpeta en su equipo local. Para obtener instrucciones, consulte Paso 1: implementar y probar localmente.

Después de extraer todos los archivos, observe que se ha creado la carpeta `code`. Para proporcionar esta carpeta a Elastic Beanstalk, debe empaquetar su contenido en un archivo `.zip`. En primer lugar, debe agregar un archivo de configuración a dicha carpeta. Su aplicación utiliza la información sobre la región y el punto de enlace para crear una tabla de DynamoDB en la región especificada y para realizar las solicitudes de operaciones subsiguientes a la tabla con el punto de enlace indicado.

1. Cambie a la carpeta en la que ha descargado la aplicación Tic-Tac-Toe.
2. En la carpeta raíz de la aplicación, cree un archivo de texto denominado `beanstalk.config` con el contenido siguiente.

```
[dynamodb]
region=<AWS region>
endpoint=<DynamoDB endpoint>
```

Por ejemplo, podría utilizar el contenido siguiente.

```
[dynamodb]
region=us-west-2
endpoint=dynamodb.us-west-2.amazonaws.com
```

Para obtener una lista de regiones disponibles, consulte Amazon DynamoDB en la Referencia general de Amazon Web Services.

Important

La región especificada en el archivo de configuración es la ubicación donde la aplicación Tic-Tac-Toe (Tres en raya) creará la tabla `Games` en DynamoDB. Debe crear la aplicación de Elastic Beanstalk que se describe en la siguiente sección en la misma región.

Note

Al crear la aplicación de Elastic Beanstalk, solicita que se lance un entorno cuyo tipo sea posible elegir el tipo de entorno. Para probar la aplicación Tic-Tac-Toe del ejemplo, puede elegir el tipo de entorno Single Instance (Instancia individual), omitir el resto e ir al paso siguiente.
No obstante, tenga en cuenta que el tipo de entorno Load balancing, autoscaling (Auto Scaling con balanceo de carga) proporciona un entorno altamente disponible y escalable, algo que debe tener en cuenta al crear e implementar otras aplicaciones. Si elige este tipo de entorno, también debe generar un UUID y agregárselo al archivo de configuración, tal y como se muestra a continuación.

```
[dynamodb]
region=us-west-2
endpoint=dynamodb.us-west-2.amazonaws.com
[flask]
secret_key= 284e784d-1a25-4a19-92bf-8eeb7a9example
```

En la comunicación cliente-servidor en la cual el servidor envía la respuesta, por motivos de seguridad el servidor envía una cookie firmada que el cliente devuelve al servidor

en la siguiente solicitud. Cuando hay un único servidor, este puede generar localmente una clave de cifrado al iniciarse. Si hay muchos servidores, todos ellos deben conocer la misma clave de cifrado; de lo contrario, no podrán leer las cookies establecidas por los demás servidores. Si se agrega `secret_key` al archivo de configuración, indica a todos los servidores que utilicen esta clave de cifrado.

3. Comprima el contenido de la carpeta raíz de la aplicación (que incluye el archivo `beanstalk.config`); por ejemplo, `TicTacToe.zip`.
4. Cargue el archivo `.zip` en un bucket de Amazon Simple Storage Service (Amazon S3). En la siguiente sección, proporcionaremos este archivo `.zip` a Elastic Beanstalk para cargarlo en el o los servidores.

   Para obtener instrucciones sobre cómo cargar un archivo en un bucket de Amazon S3, consulte Crear un Bucket y Agregar un objeto a un bucket en la Guía del usuario de Amazon Simple Storage Service.

3.4: configurar el entorno de AWS Elastic Beanstalk

En este paso, se crea una aplicación de Elastic Beanstalk, que es una colección de componentes, incluidos los entornos. En este ejemplo vamos a lanzar una instancia Amazon EC2 para implementar y ejecutar la aplicación Tic-Tac-Toe (Tres en raya).

1. Ingrese la siguiente URL personalizada con el fin de configurar una consola de Elastic Beanstalk para configurar el entorno.

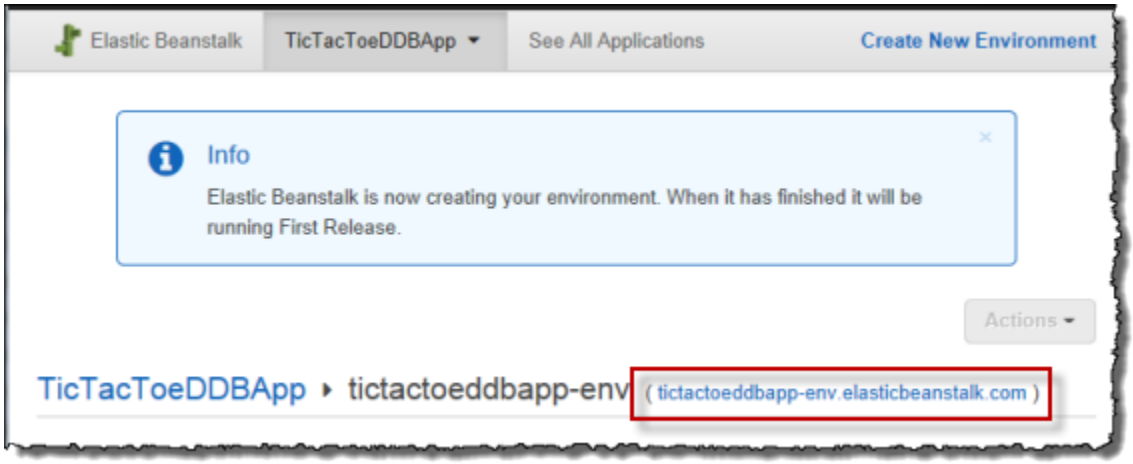
   ```
   https://console.aws.amazon.com/elasticbeanstalk/?region=<AWS-Region>#/
   newApplication
   ?applicationName=TicTacToeyour-name
   &solutionStackName=Python
   &sourceBundleUrl=https://s3.amazonaws.com/<bucket-name>/TicTacToe.zip
   &environmentType=SingleInstance
   &instanceType=t1.micro
   ```

   Para obtener más información sobre las URL personalizadas, consulte Construir una URL de lanzamiento inmediato en la Guía de desarrolladores de Elastic Beanstalk de AWS Elastic Beanstalk. Para la URL, tenga en cuenta lo siguiente:

- Debe proporcionar el nombre de una región de AWS (el mismo que ha facilitado en el archivo de configuración), un bucket de Amazon S3 y el nombre de objeto.
- Para las pruebas, la URL solicita el tipo de entorno SingleInstance y `t1.micro` como tipo de instancia.
- El nombre de la aplicación debe ser único. Así pues, en la URL anterior, sugerimos que anteponga su propio nombre a `applicationName`.

Al hacerlo, se abre la consola de Elastic Beanstalk. En algunos casos, es posible que deba iniciar sesión.

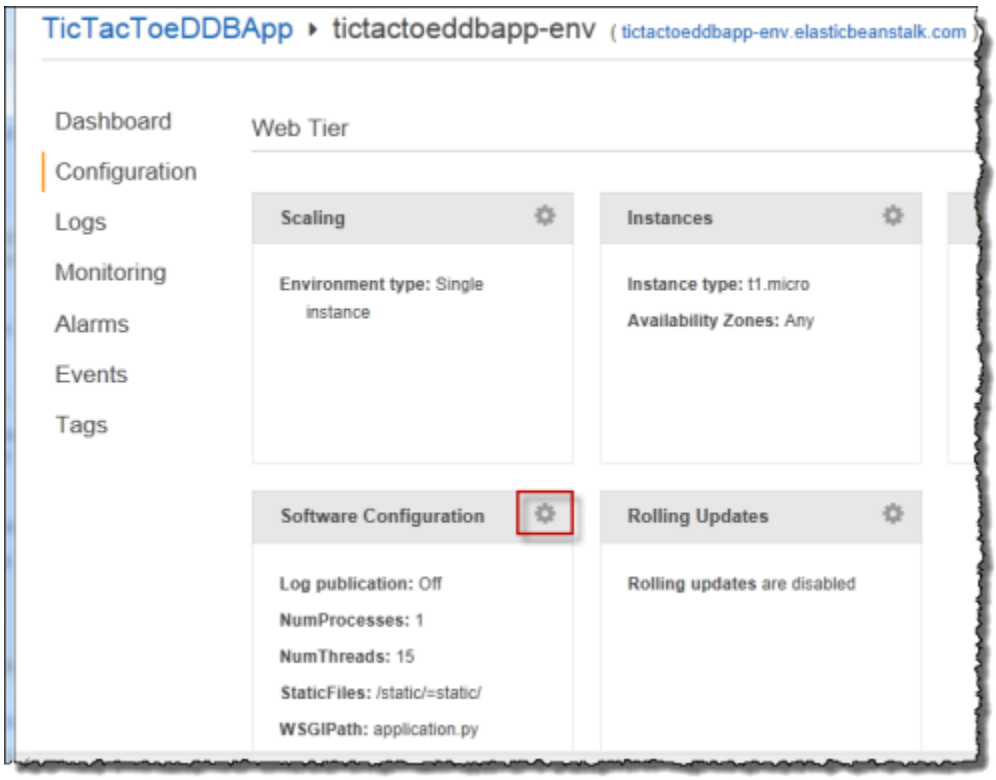
2. En la consola de Elastic Beanstalk, elija Review and Launch y, a continuación, elija Launch.
3. Anote la URL para futuras consultas. Esta URL abre la página de inicio de la aplicación Tic-Tac-Toe.



4. Configure la aplicación Tic-Tac-Toe de modo que conozca la ubicación del archivo de configuración.

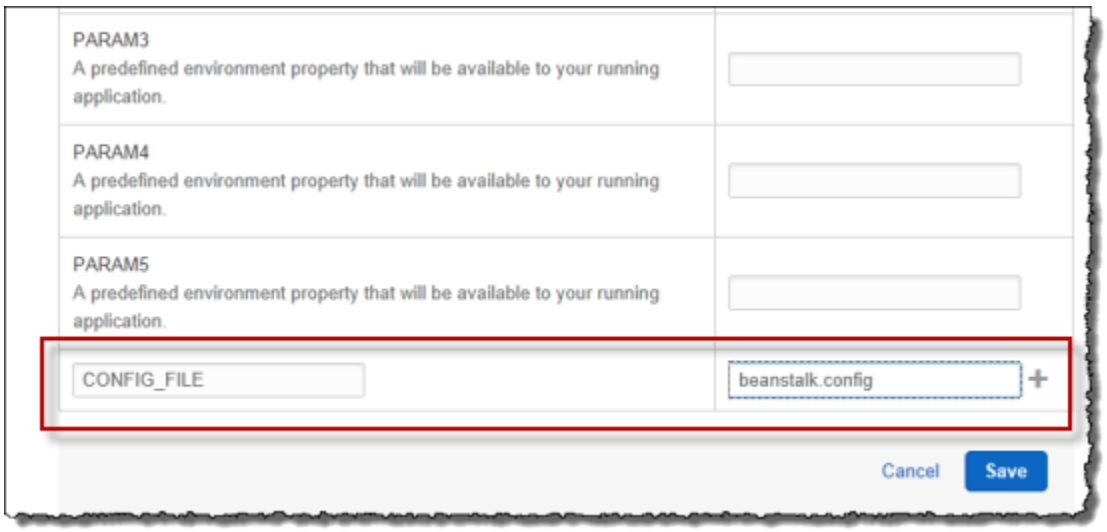
   Una vez que Elastic Beanstalk haya creado la aplicación, elija Configuration.

   a. Elija el icono con forma de engranaje que aparece junto a Software Configuration (Configuración de software), tal y como se muestra en la captura de pantalla siguiente.



b. Al final de la sección Environment Properties (Propiedades de entorno), escriba **`CONFIG_FILE`** y su valor **`beanstalk.config`**; a continuación, elija Save (Guardar).

Puede que la actualización del entorno tarde unos minutos en completarse.

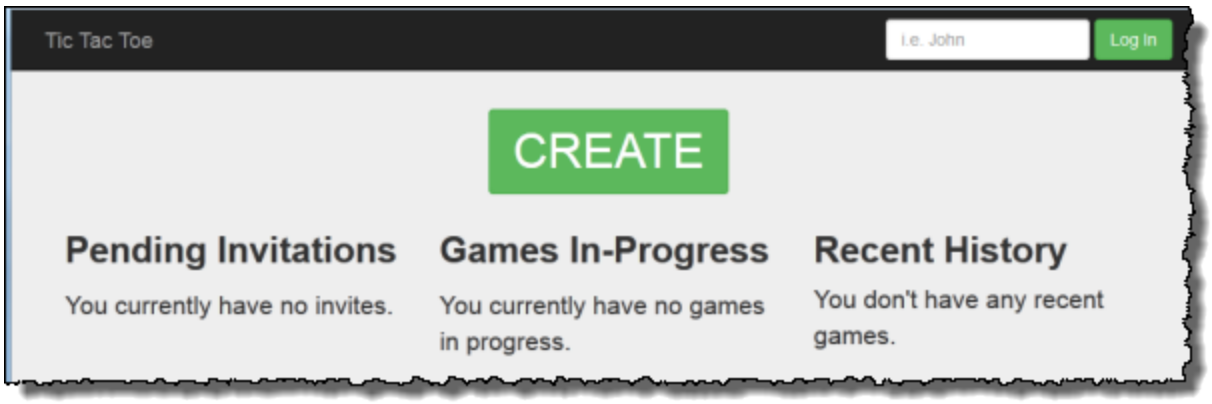


Una vez completada la actualización, ya puede jugar.

5. En el navegador, escriba la URL que copió en el paso anterior, tal y como se muestra en el siguiente ejemplo.

```
http://<pen-name>.elasticbeanstalk.com
```

Se abrirá la página de inicio de la aplicación.



6. Inicie sesión como testuser1 y elija CREATE (CREAR) para comenzar una nueva partida de tres en raya.
7. Escriba **`testuser2`** en el cuadro Choose an Opponent (Elegir un contrincante).

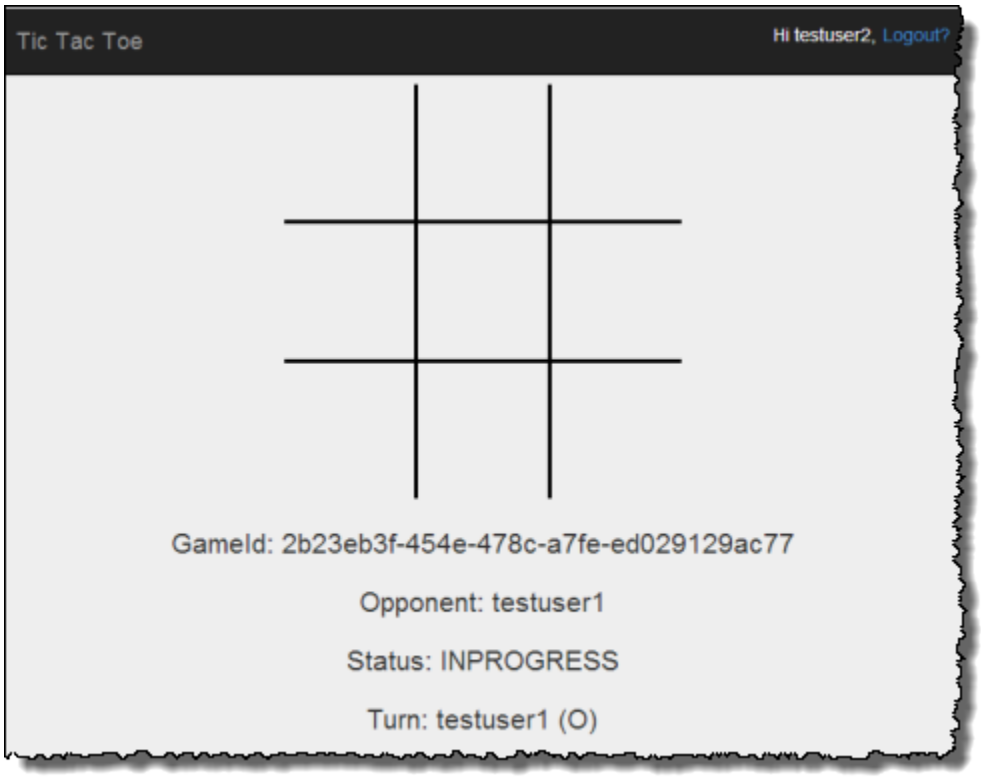
8. Abra otra ventana del navegador.

   Asegúrese de borrar todas las cookies en la ventana del navegador para no iniciar sesión con el mismo nombre de usuario.
9. Escriba la misma URL para abrir la página de inicio de la aplicación, tal y como se muestra en el ejemplo siguiente.

   ```
   http://<env-name>.elasticbeanstalk.com
   ```
10. Inicie sesión como testuser2.

11. Vaya a la invitación de testuser1 en la lista de invitaciones pendientes y elija accept (aceptar).

12. Ahora aparece la página de la partida.



Pueden jugar la partida los usuarios testuser1 y testuser2. La aplicación guardará cada jugada en el elemento correspondiente de la tabla `Games`.

Paso 4: limpie los recursos

Se han completado la implementación y las pruebas de la aplicación Tic-Tac-Toe. La aplicación abarca el desarrollo integral de aplicaciones web en Amazon DynamoDB, salvo la autenticación de usuarios. La aplicación utiliza la información de inicio de sesión de la página de inicio únicamente para agregar el nombre de un jugador al crear una partida. En una aplicación de producción, tendría que agregar el código necesario para llevar a cabo el inicio de sesión y la autenticación de los usuarios.

Si ha terminado de realizar pruebas, puede eliminar los recursos que ha creado para probar la aplicación Tic-Tac-Toe, con el fin de evitar que se le cobre algún importe.

Para eliminar los recursos que ha creado

1. Elimine la tabla `Games` que creó en DynamoDB.
2. Termine el entorno de Elastic Beanstalk para liberar las instancias de Amazon EC2.
3. Elimine el rol de IAM que ha creado.
4. Elimine el objeto que ha creado en Amazon S3.

Palabras reservadas en DynamoDB

Las siguientes palabras clave están reservadas para el uso de DynamoDB. No utilice ninguna de estas palabras como nombres de atributos en las expresiones. Esta lista no distingue entre mayúsculas y minúsculas.

Si tiene que escribir alguna expresión que contenga un nombre de atributo que entre en conflicto con una palabra reservada de DynamoDB, puede definir un nombre de atributo de expresión para usarlo en lugar de la palabra reservada. Para obtener más información, consulte Nombres de atributos de expresión (Alias) en DynamoDB.

```
ABORT
ABSOLUTE
ACTION
ADD
AFTER
AGENT
AGGREGATE
ALL
```

```
ALLOCATE
ALTER
ANALYZE
AND
ANY
ARCHIVE
ARE
ARRAY
AS
ASC
ASCII
ASENSITIVE
ASSERTION
ASYMMETRIC
AT
ATOMIC
ATTACH
ATTRIBUTE
AUTH
AUTHORIZATION
AUTHORIZE
AUTO
AVG
BACK
BACKUP
BASE
BATCH
BEFORE
BEGIN
BETWEEN
BIGINT
BINARY
BIT
BLOB
BLOCK
BOOLEAN
BOTH
BREADTH
BUCKET
BULK
BY
BYTE
CALL
CALLED
```

```
CALLING
CAPACITY
CASCADE
CASCADED
CASE
CAST
CATALOG
CHAR
CHARACTER
CHECK
CLASS
CLOB
CLOSE
CLUSTER
CLUSTERED
CLUSTERING
CLUSTERS
COALESCE
COLLATE
COLLATION
COLLECTION
COLUMN
COLUMNS
COMBINE
COMMENT
COMMIT
COMPACT
COMPILE
COMPRESS
CONDITION
CONFLICT
CONNECT
CONNECTION
CONSISTENCY
CONSISTENT
CONSTRAINT
CONSTRAINTS
CONSTRUCTOR
CONSUMED
CONTINUE
CONVERT
COPY
CORRESPONDING
COUNT
```

```
COUNTER
CREATE
CROSS
CUBE
CURRENT
CURSOR
CYCLE
DATA
DATABASE
DATE
DATETIME
DAY
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFAULT
DEFERRABLE
DEFERRED
DEFINE
DEFINED
DEFINITION
DELETE
DELIMITED
DEPTH
DEREF
DESC
DESCRIBE
DESCRIPTOR
DETACH
DETERMINISTIC
DIAGNOSTICS
DIRECTORIES
DISABLE
DISCONNECT
DISTINCT
DISTRIBUTE
DO
DOMAIN
DOUBLE
DROP
DUMP
DURATION
DYNAMIC
```

```
EACH
ELEMENT
ELSE
ELSEIF
EMPTY
ENABLE
END
EQUAL
EQUALS
ERROR
ESCAPE
ESCAPED
EVAL
EVALUATE
EXCEEDED
EXCEPT
EXCEPTION
EXCEPTIONS
EXCLUSIVE
EXEC
EXECUTE
EXISTS
EXIT
EXPLAIN
EXPLODE
EXPORT
EXPRESSION
EXTENDED
EXTERNAL
EXTRACT
FAIL
FALSE
FAMILY
FETCH
FIELDS
FILE
FILTER
FILTERING
FINAL
FINISH
FIRST
FIXED
FLATTERN
FLOAT
```

```
FOR
FORCE
FOREIGN
FORMAT
FORWARD
FOUND
FREE
FROM
FULL
FUNCTION
FUNCTIONS
GENERAL
GENERATE
GET
GLOB
GLOBAL
GO
GOTO
GRANT
GREATER
GROUP
GROUPING
HANDLER
HASH
HAVE
HAVING
HEAP
HIDDEN
HOLD
HOUR
IDENTIFIED
IDENTITY
IF
IGNORE
IMMEDIATE
IMPORT
IN
INCLUDING
INCLUSIVE
INCREMENT
INCREMENTAL
INDEX
INDEXED
INDEXES
```

```
INDICATOR
INFINITE
INITIALLY
INLINE
INNER
INNTER
INOUT
INPUT
INSENSITIVE
INSERT
INSTEAD
INT
INTEGER
INTERSECT
INTERVAL
INTO
INVALIDATE
IS
ISOLATION
ITEM
ITEMS
ITERATE
JOIN
KEY
KEYS
LAG
LANGUAGE
LARGE
LAST
LATERAL
LEAD
LEADING
LEAVE
LEFT
LENGTH
LESS
LEVEL
LIKE
LIMIT
LIMITED
LINES
LIST
LOAD
LOCAL
```

```
LOCALTIME
LOCALTIMESTAMP
LOCATION
LOCATOR
LOCK
LOCKS
LOG
LOGED
LONG
LOOP
LOWER
MAP
MATCH
MATERIALIZED
MAX
MAXLEN
MEMBER
MERGE
METHOD
METRICS
MIN
MINUS
MINUTE
MISSING
MOD
MODE
MODIFIES
MODIFY
MODULE
MONTH
MULTI
MULTISET
NAME
NAMES
NATIONAL
NATURAL
NCHAR
NCLOB
NEW
NEXT
NO
NONE
NOT
NULL
```

```
NULLIF
NUMBER
NUMERIC
OBJECT
OF
OFFLINE
OFFSET
OLD
ON
ONLINE
ONLY
OPAQUE
OPEN
OPERATOR
OPTION
OR
ORDER
ORDINALITY
OTHER
OTHERS
OUT
OUTER
OUTPUT
OVER
OVERLAPS
OVERRIDE
OWNER
PAD
PARALLEL
PARAMETER
PARAMETERS
PARTIAL
PARTITION
PARTITIONED
PARTITIONS
PATH
PERCENT
PERCENTILE
PERMISSION
PERMISSIONS
PIPE
PIPELINED
PLAN
POOL
```

```
POSITION
PRECISION
PREPARE
PRESERVE
PRIMARY
PRIOR
PRIVATE
PRIVILEGES
PROCEDURE
PROCESSED
PROJECT
PROJECTION
PROPERTY
PROVISIONING
PUBLIC
PUT
QUERY
QUIT
QUORUM
RAISE
RANDOM
RANGE
RANK
RAW
READ
READS
REAL
REBUILD
RECORD
RECURSIVE
REDUCE
REF
REFERENCE
REFERENCES
REFERENCING
REGEXP
REGION
REINDEX
RELATIVE
RELEASE
REMAINDER
RENAME
REPEAT
REPLACE
```

```
REQUEST
RESET
RESIGNAL
RESOURCE
RESPONSE
RESTORE
RESTRICT
RESULT
RETURN
RETURNING
RETURNS
REVERSE
REVOKE
RIGHT
ROLE
ROLES
ROLLBACK
ROLLUP
ROUTINE
ROW
ROWS
RULE
RULES
SAMPLE
SATISFIES
SAVE
SAVEPOINT
SCAN
SCHEMA
SCOPE
SCROLL
SEARCH
SECOND
SECTION
SEGMENT
SEGMENTS
SELECT
SELF
SEMI
SENSITIVE
SEPARATE
SEQUENCE
SERIALIZABLE
SESSION
```

```
SET
SETS
SHARD
SHARE
SHARED
SHORT
SHOW
SIGNAL
SIMILAR
SIZE
SKEWED
SMALLINT
SNAPSHOT
SOME
SOURCE
SPACE
SPACES
SPARSE
SPECIFIC
SPECIFICTYPE
SPLIT
SQL
SQLCODE
SQLERROR
SQLEXCEPTION
SQLSTATE
SQLWARNING
START
STATE
STATIC
STATUS
STORAGE
STORE
STORED
STREAM
STRING
STRUCT
STYLE
SUB
SUBMULTISET
SUBPARTITION
SUBSTRING
SUBTYPE
SUM
```

```
SUPER
SYMMETRIC
SYNONYM
SYSTEM
TABLE
TABLESAMPLE
TEMP
TEMPORARY
TERMINATED
TEXT
THAN
THEN
THROUGHPUT
TIME
TIMESTAMP
TIMEZONE
TINYINT
TO
TOKEN
TOTAL
TOUCH
TRAILING
TRANSACTION
TRANSFORM
TRANSLATE
TRANSLATION
TREAT
TRIGGER
TRIM
TRUE
TRUNCATE
TTL
TUPLE
TYPE
UNDER
UNDO
UNION
UNIQUE
UNIT
UNKNOWN
UNLOGGED
UNNEST
UNPROCESSED
UNSIGNED
```

```
UNTIL
UPDATE
UPPER
URL
USAGE
USE
USER
USERS
USING
UUID
VACUUM
VALUE
VALUED
VALUES
VARCHAR
VARIABLE
VARIANCE
VARINT
VARYING
VIEW
VIEWS
VIRTUAL
VOID
WAIT
WHEN
WHENEVER
WHERE
WHILE
WINDOW
WITH
WITHIN
WITHOUT
WORK
WRAPPED
WRITE
YEAR
ZONE
```

AWSEjemplos de SDK de para Java 1.x

Esta sección contiene código de ejemplo para aplicaciones DAX que utilizan SDK for Java 1.x.

Temas

- Uso de DAX con SDK de AWS para Java 1.x
- Modificación de una aplicación SDK para Java 1.x existente para que use DAX
- Consulta de índices secundarios globales con SDK para Java 1.x

Uso de DAX con SDK de AWS para Java 1.x

Siga este procedimiento para ejecutar el ejemplo de Java para Amazon DynamoDB Accelerator (DAX) en su instancia de Amazon EC2.

> **Note**
>
> Estas instrucciones son para aplicaciones que utilizan SDK de AWS para Java 1.x. Para aplicaciones que utilizan SDK de AWS para Java 2.x, consulte Java y DAX.

Para ejecutar el ejemplo de Java para DAX

1. Instale el kit de desarrollo de Java (JDK).

```
sudo yum install -y java-devel
```

2. Descargue el archivo del AWS SDK para Java (archivo `.zip`) y, a continuación, extráigalo.

```
wget http://sdk-for-java.amazonwebservices.com/latest/aws-java-sdk.zip

unzip aws-java-sdk.zip
```

3. Descargue en la última versión del cliente Java de DAX; (archivo `.jar`):

```
wget http://dax-sdk.s3-website-us-west-2.amazonaws.com/java/DaxJavaClient-
latest.jar
```

> **Note**
>
> El cliente del SDK de DAX para Java está disponible en Apache Maven. Para obtener más información, consulte Uso del cliente como dependencia de Apache Maven.

4. Establezca la variable `CLASSPATH`. En este ejemplo, sustituya *`sdkVersion`* por el número de versión real de AWS SDK para Java, (por ejemplo: `1.11.112`).

```
export SDKVERSION=sdkVersion

export CLASSPATH=$(pwd)/TryDax/java:$(pwd)/DaxJavaClient-latest.jar:$(pwd)/
aws-java-sdk-$SDKVERSION/lib/aws-java-sdk-$SDKVERSION.jar:$(pwd)/aws-java-sdk-
$SDKVERSION/third-party/lib/*
```

5. Descargue el código fuente del programa de ejemplo (archivo `.zip`).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

   Cuando haya terminado la descarga, extraiga los archivos de código fuente.

```
unzip TryDax.zip
```

6. Navegue hasta el directorio de código Java y compile el código de la siguiente manera.

```
cd TryDax/java/
javac TryDax*.java
```

7. Ejecute el programa.

```
java TryDax
```

   Debería ver un resultado similar a este.

```
Creating a DynamoDB client

Attempting to create table; please wait...
Successfully created table.  Table status: ACTIVE
Writing data to the table...
Writing 10 items for partition key: 1
Writing 10 items for partition key: 2
Writing 10 items for partition key: 3
Writing 10 items for partition key: 4
Writing 10 items for partition key: 5
Writing 10 items for partition key: 6
Writing 10 items for partition key: 7
```

```
Writing 10 items for partition key: 8
Writing 10 items for partition key: 9
Writing 10 items for partition key: 10

Running GetItem, Scan, and Query tests...
First iteration of each test will result in cache misses
Next iterations are cache hits

GetItem test - partition key 1 and sort keys 1-10
 Total time: 136.681 ms - Avg time: 13.668 ms
 Total time: 122.632 ms - Avg time: 12.263 ms
 Total time: 167.762 ms - Avg time: 16.776 ms
 Total time: 108.130 ms - Avg time: 10.813 ms
 Total time: 137.890 ms - Avg time: 13.789 ms
Query test - partition key 5 and sort keys between 2 and 9
 Total time: 13.560 ms - Avg time: 2.712 ms
 Total time: 11.339 ms - Avg time: 2.268 ms
 Total time: 7.809 ms - Avg time: 1.562 ms
 Total time: 10.736 ms - Avg time: 2.147 ms
 Total time: 12.122 ms - Avg time: 2.424 ms
Scan test - all items in the table
 Total time: 58.952 ms - Avg time: 11.790 ms
 Total time: 25.507 ms - Avg time: 5.101 ms
 Total time: 37.660 ms - Avg time: 7.532 ms
 Total time: 26.781 ms - Avg time: 5.356 ms
 Total time: 46.076 ms - Avg time: 9.215 ms

Attempting to delete table; please wait...
Successfully deleted table.
```

Tome nota de la información de tiempo; es decir, del número de milisegundos necesarios para realizar las pruebas de `GetItem`, `Query` y `Scan`.

8. En el paso anterior, ha ejecutado el programa en el punto de enlace de DynamoDB. Ahora, ejecute el programa de nuevo, pero, esta vez, las operaciones `GetItem`, `Query` y `Scan` se procesan en el clúster de DAX.

   Para determinar el punto de enlace del clúster de DAX, elija una de las opciones siguientes:

   - En la consola de DynamoDB: elija su clúster de DAX. El punto de enlace del clúster se muestra en la consola, como en el siguiente ejemplo.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- En la AWS CLI: ingrese el siguiente comando.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

El punto de enlace del clúster se muestra en el resultado, como en el siguiente ejemplo.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

Ahora, vuelva a ejecutar el programa, pero, esta vez, especifique el punto de enlace del clúster como parámetro en la línea de comandos.

```
java TryDax dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Fíjese en el resto del resultado y tome nota de la información sobre tiempos. Los tiempos transcurridos para las operaciones `GetItem`, `Query` y `Scan` deberían ser significativamente menores con DAX que con DynamoDB.

Para obtener más información sobre este programa, consulte las siguientes secciones:

- TryDax.java
- TryDaxHelper.java
- TryDaxTests.java

Uso del cliente como dependencia de Apache Maven

Siga estos pasos para utilizar el cliente para el SDK de DAX para Java en su aplicación como una dependencia.

Para usar el cliente como una dependencia de Maven

1. Descargue e instale Apache Maven. Para obtener más información, consulte Downloading Apache Maven e Installing Apache Maven.
2. Agregue la dependencia de Maven del cliente al archivo POM (Project Object Model) de la aplicación. En este ejemplo, sustituya `x.x.x.x` por el número de versión real del cliente (por ejemplo: `1.0.200704.0`).

```
<!--Dependency:-->
<dependencies>
    <dependency>
     <groupId>com.amazonaws</groupId>
     <artifactId>amazon-dax-client</artifactId>
     <version>x.x.x.x</version>
    </dependency>
</dependencies>
```

TryDax.java

El archivo `TryDax.java` contiene el método `main`. Si ejecuta el programa sin parámetros de línea de comandos, se crea un cliente de Amazon DynamoDB que se utiliza para todas las operaciones de la API. Si especifica el punto de enlace de un clúster de DynamoDB Accelerator (DAX) en la línea de comandos, el programa crea también un cliente de DAX y lo utiliza para las operaciones `GetItem`, `Query` y `Scan`.

Puede modificar el programa de varias maneras:

- Utilizar el cliente de DAX en lugar del cliente de DynamoDB. Para obtener más información, consulte Java y DAX.
- Elegir otro nombre para la tabla de prueba.
- Cambiar los parámetros de `helper.writeData` para modificar el número de elementos que se escriben. El segundo parámetro es el número de claves de partición y el tercero, el número de claves de ordenación. De forma predeterminada, el programa usa entre 1 y 10 valores de clave de partición y entre 1 y 10 valores de clave de ordenación, lo que equivale a un total de 100 elementos escritos en la tabla. Para obtener más información, consulte TryDaxHelper.java.
- Modificar el número de pruebas de `GetItem`, `Query` y `Scan`, así como sus parámetros.

- Marcar como comentarios las líneas que contienen `helper.createTable` y `helper.deleteTable`, sin no desea crear y eliminar la tabla cada vez que ejecute el programa.

> **Note**
>
> Para ejecutar este programa, puede configurar Maven para usar el cliente para el SDK de DAX para Java y AWS SDK para Java como dependencias. Para obtener más información, consulte Uso del cliente como dependencia de Apache Maven.
> Como alternativa, puede descargar e incluir el cliente Java de DAX y el AWS SDK para Java en su classpath. Consulte Java y DAX para ver un ejemplo de cómo establecer la variable `CLASSPATH`.

```
public class TryDax {

    public static void main(String[] args) throws Exception {

        TryDaxHelper helper = new TryDaxHelper();
        TryDaxTests tests = new TryDaxTests();

        DynamoDB ddbClient = helper.getDynamoDBClient();
        DynamoDB daxClient = null;
        if (args.length >= 1) {
            daxClient = helper.getDaxClient(args[0]);
        }

        String tableName = "TryDaxTable";

        System.out.println("Creating table...");
        helper.createTable(tableName, ddbClient);
        System.out.println("Populating table...");
        helper.writeData(tableName, ddbClient, 10, 10);

        DynamoDB testClient = null;
        if (daxClient != null) {
            testClient = daxClient;
        } else {
            testClient = ddbClient;
        }
```

```
        System.out.println("Running GetItem, Scan, and Query tests...");
        System.out.println("First iteration of each test will result in cache misses");
        System.out.println("Next iterations are cache hits\n");

        // GetItem
        tests.getItemTest(tableName, testClient, 1, 10, 5);

        // Query
        tests.queryTest(tableName, testClient, 5, 2, 9, 5);

        // Scan
        tests.scanTest(tableName, testClient, 5);

        helper.deleteTable(tableName, ddbClient);
    }

}
```

TryDaxHelper.java

El archivo `TryDaxHelper.java` contiene métodos de utilidad.

Los métodos `getDynamoDBClient` y `getDaxClient` proporcionan clientes de Amazon DynamoDB y DynamoDB Accelerator (DAX). Para las operaciones del plano de control (`CreateTable`, `DeleteTable`) y las operaciones de escritura, el programa utiliza el cliente de DynamoDB. Si especifica el punto de enlace de un clúster de DAX, el programa principal crea un cliente de DAX para llevar a cabo las operaciones de lectura (`GetItem`, `Query`, `Scan`).

Los demás métodos de `TryDaxHelper` (`createTable`, `writeData` y `deleteTable`) se utilizan para configurar y eliminar la tabla de DynamoDB y sus datos.

Puede modificar el programa de varias maneras:

- Utilizar unos ajustes de desempeño provisionado diferentes para la tabla.
- Modificar el tamaño de cada elemento escrito (consulte la variable `stringSize` del método `writeData`).
- Modificar el número de pruebas de `GetItem`, `Query` y `Scan` y sus parámetros.
- Marcar como comentarios las líneas que contienen `helper.CreateTable` y `helper.DeleteTable`, sin no desea crear y eliminar la tabla cada vez que ejecute el programa.

> **Note**
>
> Para ejecutar este programa, puede configurar Maven para usar el cliente para el SDK de DAX para Java y AWS SDK para Java como dependencias. Para obtener más información, consulte Uso del cliente como dependencia de Apache Maven.
> O, puede descargar e incluir el cliente Java de DAX y el AWS SDK para Java en su classpath. Consulte Java y DAX para ver un ejemplo de cómo establecer la variable `CLASSPATH`.

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.model.AttributeDefinition;
import com.amazonaws.services.dynamodbv2.model.KeySchemaElement;
import com.amazonaws.services.dynamodbv2.model.KeyType;
import com.amazonaws.services.dynamodbv2.model.ProvisionedThroughput;
import com.amazonaws.services.dynamodbv2.model.ScalarAttributeType;
import com.amazonaws.util.EC2MetadataUtils;

public class TryDaxHelper {

    private static final String region = EC2MetadataUtils.getEC2InstanceRegion();

    DynamoDB getDynamoDBClient() {
        System.out.println("Creating a DynamoDB client");
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard()
                .withRegion(region)
                .build();
        return new DynamoDB(client);
    }

    DynamoDB getDaxClient(String daxEndpoint) {
        System.out.println("Creating a DAX client with cluster endpoint " +
 daxEndpoint);
        AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
        daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
```

```
        AmazonDynamoDB client = daxClientBuilder.build();
        return new DynamoDB(client);
    }

    void createTable(String tableName, DynamoDB client) {
        Table table = client.getTable(tableName);
        try {
            System.out.println("Attempting to create table; please wait...");

            table = client.createTable(tableName,
                    Arrays.asList(
                            new KeySchemaElement("pk", KeyType.HASH), // Partition key
                            new KeySchemaElement("sk", KeyType.RANGE)), // Sort key
                    Arrays.asList(
                            new AttributeDefinition("pk", ScalarAttributeType.N),
                            new AttributeDefinition("sk", ScalarAttributeType.N)),
                    new ProvisionedThroughput(10L, 10L));
            table.waitForActive();
            System.out.println("Successfully created table.  Table status: " +
                    table.getDescription().getTableStatus());

        } catch (Exception e) {
            System.err.println("Unable to create table: ");
            e.printStackTrace();
        }
    }

    void writeData(String tableName, DynamoDB client, int pkmax, int skmax) {
        Table table = client.getTable(tableName);
        System.out.println("Writing data to the table...");

        int stringSize = 1000;
        StringBuilder sb = new StringBuilder(stringSize);
        for (int i = 0; i < stringSize; i++) {
            sb.append('X');
        }
        String someData = sb.toString();

        try {
            for (Integer ipk = 1; ipk <= pkmax; ipk++) {
                System.out.println(("Writing " + skmax + " items for partition key: " +
ipk));
                for (Integer isk = 1; isk <= skmax; isk++) {
                    table.putItem(new Item()
```

```
                            .withPrimaryKey("pk", ipk, "sk", isk)
                            .withString("someData", someData));
                }
            }
        } catch (Exception e) {
            System.err.println("Unable to write item:");
            e.printStackTrace();
        }
    }

    void deleteTable(String tableName, DynamoDB client) {
        Table table = client.getTable(tableName);
        try {
            System.out.println("\nAttempting to delete table; please wait...");
            table.delete();
            table.waitForDelete();
            System.out.println("Successfully deleted table.");

        } catch (Exception e) {
            System.err.println("Unable to delete table: ");
            e.printStackTrace();
        }
    }

}
```

TryDaxTests.java

El archivo `TryDaxTests.java` contiene métodos que llevan a cabo operaciones de lectura en una tabla de prueba en Amazon DynamoDB . Estos métodos no tienen en cuenta cómo acceder a los datos (mediante el cliente de DynamoDB o de DAX), por lo que no es necesario modificar la lógica de la aplicación.

Puede modificar el programa de varias maneras:

- Modificar el método `queryTest` de modo que use otra expresión `KeyConditionExpression`.
- Agregar un filtro `ScanFilter` al método `scanTest` para que solamente se devuelvan algunos de los elementos.

> **Note**
>
> Para ejecutar este programa, puede configurar Maven para usar el cliente para el SDK de DAX para Java y AWS SDK para Java como dependencias. Para obtener más información, consulte Uso del cliente como dependencia de Apache Maven.
> O, puede descargar e incluir el cliente Java de DAX y el AWS SDK para Java en su classpath. Consulte Java y DAX para ver un ejemplo de cómo establecer la variable `CLASSPATH`.

```
import java.util.Iterator;

import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.ItemCollection;
import com.amazonaws.services.dynamodbv2.document.QueryOutcome;
import com.amazonaws.services.dynamodbv2.document.ScanOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.dynamodbv2.document.spec.QuerySpec;

public class TryDaxTests {

    void getItemTest(String tableName, DynamoDB client, int pk, int sk, int iterations)
 {
        long startTime, endTime;
        System.out.println("GetItem test - partition key " + pk + " and sort keys 1-" +
 sk);
        Table table = client.getTable(tableName);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            try {
                for (Integer ipk = 1; ipk <= pk; ipk++) {
                    for (Integer isk = 1; isk <= sk; isk++) {
                        table.getItem("pk", ipk, "sk", isk);
                    }
                }
            } catch (Exception e) {
                System.err.println("Unable to get item:");
                e.printStackTrace();
            }
```

```
            endTime = System.nanoTime();
            printTime(startTime, endTime, pk * sk);
        }
    }

    void queryTest(String tableName, DynamoDB client, int pk, int sk1, int sk2, int
iterations) {
        long startTime, endTime;
        System.out.println("Query test - partition key " + pk + " and sort keys between
" + sk1 + " and " + sk2);
        Table table = client.getTable(tableName);

        HashMap<String, Object> valueMap = new HashMap<String, Object>();
        valueMap.put(":pkval", pk);
        valueMap.put(":skval1", sk1);
        valueMap.put(":skval2", sk2);

        QuerySpec spec = new QuerySpec()
                .withKeyConditionExpression("pk = :pkval and sk between :skval1
and :skval2")
                .withValueMap(valueMap);

        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            ItemCollection<QueryOutcome> items = table.query(spec);

            try {
                Iterator<Item> iter = items.iterator();
                while (iter.hasNext()) {
                    iter.next();
                }
            } catch (Exception e) {
                System.err.println("Unable to query table:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, iterations);
        }
    }

    void scanTest(String tableName, DynamoDB client, int iterations) {
        long startTime, endTime;
        System.out.println("Scan test - all items in the table");
        Table table = client.getTable(tableName);
```

```
        for (int i = 0; i < iterations; i++) {
            startTime = System.nanoTime();
            ItemCollection<ScanOutcome> items = table.scan();
            try {

                Iterator<Item> iter = items.iterator();
                while (iter.hasNext()) {
                    iter.next();
                }
            } catch (Exception e) {
                System.err.println("Unable to scan table:");
                e.printStackTrace();
            }
            endTime = System.nanoTime();
            printTime(startTime, endTime, iterations);
        }
    }

    public void printTime(long startTime, long endTime, int iterations) {
        System.out.format("\tTotal time: %.3f ms - ", (endTime - startTime) /
 (1000000.0));
        System.out.format("Avg time: %.3f ms\n", (endTime - startTime) / (iterations *
 1000000.0));
    }
}
```

Modificación de una aplicación SDK para Java 1.x existente para que use DAX

Si ya dispone de una aplicación de Java que utiliza Amazon DynamoDB, deberá modificarla para que pueda acceder al clúster de DynamoDB Accelerator (DAX). No tiene que volver a escribir toda la aplicación porque el cliente Java de DAX es similar al cliente de bajo nivel de DynamoDB que se incluye en el AWS SDK para Java.

> **Note**
>
> Estas instrucciones son para aplicaciones que utilizan SDK de AWS para Java 1.x. Para aplicaciones que utilizan SDK de AWS para Java 2.x, consulte Modificación de una aplicación existente para que use DAX.

Supongamos que tiene una tabla de DynamoDB denominada `Music`. La clave de partición de la tabla es `Artist` y la de ordenación `SongTitle`. El siguiente programa lee un elemento directamente desde la tabla `Music`.

```
import java.util.HashMap;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class GetMusicItem {

    public static void main(String[] args) throws Exception {

        // Create a DynamoDB client
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();

        HashMap<String, AttributeValue> key = new HashMap<String, AttributeValue>();
        key.put("Artist", new AttributeValue().withS("No One You Know"));
        key.put("SongTitle", new AttributeValue().withS("Scared of My Shadow"));

        GetItemRequest request = new GetItemRequest()
            .withTableName("Music").withKey(key);

        try {
            System.out.println("Attempting to read the item...");
            GetItemResult result = client.getItem(request);
            System.out.println("GetItem succeeded: " + result);

        } catch (Exception e) {
            System.err.println("Unable to read item");
            System.err.println(e.getMessage());
        }
    }
}
```

Para modificar el programa, se sustituye el cliente de DynamoDB por un cliente de DAX.

```
import java.util.HashMap;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
```

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class GetMusicItem {

    public static void main(String[] args) throws Exception {

    //Create a DAX client

    AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
    daxClientBuilder.withRegion("us-
east-1").withEndpointConfiguration("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com
    AmazonDynamoDB client = daxClientBuilder.build();


       /*
       ** ...
       ** Remaining code omitted (it is identical)
       ** ...
       */

    }
}
```

Uso de la API de documentos de DynamoDB

El AWS SDK para Java proporciona una interfaz de documentos para DynamoDB. La API de documentos actúa como encapsulador del cliente de bajo nivel de DynamoDB. Para obtener más información, consulte Interfaces de documentos.

La interfaz de documentos también se pueden utilizar con el cliente de bajo nivel de DAX, tal y como se muestra en el siguiente ejemplo.

```
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.GetItemOutcome;
import com.amazonaws.services.dynamodbv2.document.Table;

public class GetMusicItemWithDocumentApi {
```

```
    public static void main(String[] args) throws Exception {

        //Create a DAX client

        AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();
        daxClientBuilder.withRegion("us-
east-1").withEndpointConfiguration("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com
        AmazonDynamoDB client = daxClientBuilder.build();

        // Document client wrapper
        DynamoDB docClient = new DynamoDB(client);

        Table table = docClient.getTable("Music");

        try {
            System.out.println("Attempting to read the item...");
            GetItemOutcome outcome = table.tgetItemOutcome(
                "Artist", "No One You Know",
                "SongTitle", "Scared of My Shadow");
            System.out.println(outcome.getItem());
            System.out.println("GetItem succeeded: " + outcome);
        } catch (Exception e) {
            System.err.println("Unable to read item");
            System.err.println(e.getMessage());
        }

    }
}
```

Cliente asincrónico de DAX

El `AmazonDaxClient` es sincrónico. Para las operaciones de la API de DAX cuya ejecución dura mucho, tales como una operación de `Scan` en una tabla grande, esto puede bloquear la ejecución del programa hasta que se haya completado la operación. Si el programa necesita realizar otros trabajos mientras que la operación del API de DAX está en curso, puede utilizar `ClusterDaxAsyncClient` en su lugar.

El programa siguiente ilustra cómo utilizar `ClusterDaxAsyncClient` con `Future` de Java para implementar una solución que no genere ningún bloqueo.

```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;
```

```
import com.amazon.dax.client.dynamodbv2.ClientConfig;
import com.amazon.dax.client.dynamodbv2.ClusterDaxAsyncClient;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.handlers.AsyncHandler;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBAsync;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.GetItemRequest;
import com.amazonaws.services.dynamodbv2.model.GetItemResult;

public class DaxAsyncClientDemo {
 public static void main(String[] args) throws Exception {

  ClientConfig daxConfig = new ClientConfig().withCredentialsProvider(new
 ProfileCredentialsProvider())
    .withEndpoints("mydaxcluster.2cmrwl.clustercfg.dax.use1.cache.amazonaws.com:8111");

  AmazonDynamoDBAsync client = new ClusterDaxAsyncClient(daxConfig);

  HashMap<String, AttributeValue> key = new HashMap<String, AttributeValue>();
  key.put("Artist", new AttributeValue().withS("No One You Know"));
  key.put("SongTitle", new AttributeValue().withS("Scared of My Shadow"));

  GetItemRequest request = new GetItemRequest()
    .withTableName("Music").withKey(key);

  // Java Futures
  Future<GetItemResult> call = client.getItemAsync(request);
  while (!call.isDone()) {
   // Do other processing while you're waiting for the response
   System.out.println("Doing something else for a few seconds...");
   Thread.sleep(3000);
  }
  // The results should be ready by now

  try {
   call.get();

  } catch (ExecutionException ee) {
   // Futures always wrap errors as an ExecutionException.
   // The *real* exception is stored as the cause of the
   // ExecutionException
   Throwable exception = ee.getCause();
   System.out.println("Error getting item: " + exception.getMessage());
```

```
  }

  // Async callbacks
  call = client.getItemAsync(request, new AsyncHandler<GetItemRequest, GetItemResult>()
 {

   @Override
   public void onSuccess(GetItemRequest request, GetItemResult getItemResult) {
    System.out.println("Result: " + getItemResult);
   }

   @Override
   public void onError(Exception e) {
    System.out.println("Unable to read item");
    System.err.println(e.getMessage());
    // Callers can also test if exception is an instance of
    // AmazonServiceException or AmazonClientException and cast
    // it to get additional information
   }

  });
  call.get();

 }
}
```

Consulta de índices secundarios globales con SDK para Java 1.x

Puede utilizar Amazon DynamoDB Accelerator (DAX) para consultar Índices secundarios globales usando las Interfaces de programación de DynamoDB.

El siguiente ejemplo demuestra cómo usar DAX para consultar el índice secundario global `CreateDateIndex` creado en Ejemplo: índices secundarios globales usando la API de documentos de AWS SDK para Java.

La clase `DAXClient` crea instancias de los objetos de cliente que se necesitan para interactuar con las interfaces de programación de DynamoDB.

```
import com.amazon.dax.client.dynamodbv2.AmazonDaxClientBuilder;
import software.amazon.dynamodb.datamodeling.DynamoDBMapper;
import software.amazon.dynamodb.document.DynamoDB;
import com.amazonaws.util.EC2MetadataUtils;
```

```
import software.amazon.dynamodb.AmazonDynamoDB;

public class DaxClient {

 private static final String region = EC2MetadataUtils.getEC2InstanceRegion();

 DynamoDB getDaxDocClient(String daxEndpoint) {
  System.out.println("Creating a DAX client with cluster endpoint " + daxEndpoint);
  AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();

  daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
  AmazonDynamoDB client = daxClientBuilder.build();

  return new DynamoDB(client);
 }

 DynamoDBMapper getDaxMapperClient(String daxEndpoint) {
  System.out.println("Creating a DAX client with cluster endpoint " + daxEndpoint);
  AmazonDaxClientBuilder daxClientBuilder = AmazonDaxClientBuilder.standard();

  daxClientBuilder.withRegion(region).withEndpointConfiguration(daxEndpoint);
  AmazonDynamoDB client = daxClientBuilder.build();

  return new DynamoDBMapper(client);
 }
}
```

Puede consultar un índice secundario global en de las siguientes maneras:

- Use el método `queryIndex` de la clase `QueryIndexDax` definida en el ejemplo siguiente. `QueryIndexDax` toma como parámetro el objeto de cliente devuelto por el método `getDaxDocClient` para la clase `DaxClient`.
- Si usa la interfaz de persistencia de objetos, utilice el método `queryIndexMapper` de la clase `QueryIndexDax` definida en el ejemplo siguiente. `queryIndexMapper` toma como parámetro el objeto de cliente devuelto por el método `getDaxMapperClient` definido para la clase `DaxClient`.

```
import java.util.Iterator;
import software.amazon.dynamodb.datamodeling.DynamoDBMapper;
import java.util.List;
import software.amazon.dynamodb.datamodeling.DynamoDBQueryExpression;
```

```
import software.amazon.dynamodb.model.AttributeValue;
import java.util.HashMap;
import software.amazon.dynamodb.document.Item;
import software.amazon.dynamodb.document.utils.ValueMap;
import software.amazon.dynamodb.document.spec.QuerySpec;
import software.amazon.dynamodb.document.QueryOutcome;
import software.amazon.dynamodb.document.ItemCollection;
import software.amazon.dynamodb.document.Index;
import software.amazon.dynamodb.document.Table;
import software.amazon.dynamodb.document.DynamoDB;

public class QueryIndexDax {

 //This is used to query Index using the low-level interface.
 public static void queryIndex(DynamoDB client, String tableName, String indexName) {
  Table table = client.getTable(tableName);

  System.out.println("\n*******************************************************
\n");
  System.out.print("Querying index " + indexName + "...");

  Index index = table.getIndex(indexName);

  ItemCollection<QueryOutcome> items = null;

  QuerySpec querySpec = new QuerySpec();

  if (indexName == "CreateDateIndex") {
   System.out.println("Issues filed on 2013-11-01");
   querySpec.withKeyConditionExpression("CreateDate = :v_date and
 begins_with(IssueId, :v_issue)")
     .withValueMap(new ValueMap().withString(":v_date",
 "2013-11-01").withString(":v_issue", "A-"));
   items = index.query(querySpec);
  } else {
   System.out.println("\nNo valid index name provided");
   return;
  }

  Iterator<Item> iterator = items.iterator();

  System.out.println("Query: printing results...");

  while (iterator.hasNext()) {
```

```
    System.out.println(iterator.next().toJSONPretty());
   }

  }

  //This is used to query Index using the high-level mapper interface.
  public static void queryIndexMapper(DynamoDBMapper mapper, String tableName, String
  indexName) {
   HashMap<String, AttributeValue> eav = new HashMap<String, AttributeValue>();
   eav.put(":v_date", new AttributeValue().withS("2013-11-01"));
   eav.put(":v_issue", new AttributeValue().withS("A-"));
   DynamoDBQueryExpression<CreateDate> queryExpression = new
  DynamoDBQueryExpression<CreateDate>()
      .withIndexName("CreateDateIndex").withConsistentRead(false)
      .withKeyConditionExpression("CreateDate = :v_date and
  begins_with(IssueId, :v_issue)")
      .withExpressionAttributeValues(eav);

   List<CreateDate> items = mapper.query(CreateDate.class, queryExpression);
   Iterator<CreateDate> iterator = items.iterator();

   System.out.println("Query: printing results...");

   while (iterator.hasNext()) {
    CreateDate iterObj = iterator.next();
    System.out.println(iterObj.getCreateDate());
    System.out.println(iterObj.getIssueId());
   }
  }
}
```

La definición de clase siguiente representa la tabla Issues y se usa en el método `queryIndexMapper`.

```
import software.amazon.dynamodb.datamodeling.DynamoDBTable;
import software.amazon.dynamodb.datamodeling.DynamoDBIndexHashKey;
import software.amazon.dynamodb.datamodeling.DynamoDBIndexRangeKey;
import software.amazon.dynamodb.datamodeling.DynamoDBHashKey;

@DynamoDBTable(tableName = "Issues")
public class CreateDate {
 private String createDate;
 @DynamoDBHashKey(attributeName = "IssueId")
```

```
 private String issueId;

 @DynamoDBIndexHashKey(globalSecondaryIndexName = "CreateDateIndex", attributeName =
 "CreateDate")
 public String getCreateDate() {
  return createDate;
 }

 public void setCreateDate(String createDate) {
  this.createDate = createDate;
 }

 @DynamoDBIndexRangeKey(globalSecondaryIndexName = "CreateDateIndex", attributeName =
 "IssueId")
 public String getIssueId() {
  return issueId;
 }

 public void setIssueId(String issueId) {
  this.issueId = issueId;
 }
}
```

Ejemplos de AWS SDK para Go 1.x

Esta sección contiene código de ejemplo para aplicaciones DAX que utilizan Go 1.x.

Temas

- SDK para Go de DAX

SDK para Go de DAX

Siga este procedimiento para ejecutar la aplicación de ejemplo de SDK para Go de Amazon DynamoDB Accelerator (DAX) en su instancia de Amazon EC2.

Para ejecutar la muestra de SDK para Go para DAX

1. Configure SDK para Go en su instancia de Amazon EC2:

    a. Instale el lenguaje de programación Go (`Golang`).

```
sudo yum install -y golang
```

b. Compruebe que Golang está instalado y se ejecuta correctamente.

```
go version
```

Debería aparecer un mensaje como este.

```
go version go1.15.5 linux/amd64
```

Las instrucciones restantes se basan en el soporte del módulo, que se convirtió en el predeterminado la versión 1.13 de Go.

2. Instale la aplicación Golang de ejemplo.

```
go get github.com/aws-samples/aws-dax-go-sample
```

3. Ejecute los siguientes programas de Golang. El primer programa crea una tabla de DynamoDB denominada `TryDaxGoTable`. El segundo programa escribe datos en la tabla.

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command create-table
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command put-item
```

4. Ejecute los siguientes programas de Golang.

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command get-item
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command query
```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command scan
```

Tome nota de la información de tiempo; es decir, del número de milisegundos necesarios para realizar las pruebas de `GetItem`, `Query` y `Scan`.

5. En el paso anterior, ha ejecutado los programas en el punto de enlace de DynamoDB. Ahora, ejecute los programas de nuevo, pero, esta vez, las operaciones `GetItem`, `Query` y `Scan` se procesan en el clúster de DAX.

   Para determinar el punto de enlace del clúster de DAX, elija una de las opciones siguientes:

   - En la consola de DynamoDB: elija su clúster de DAX. El punto de enlace del clúster se muestra en la consola, como en el siguiente ejemplo.

     ```
     dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
     ```

   - En la AWS CLI: ingrese el siguiente comando.

     ```
     aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
     ```

     El punto de enlace del clúster se muestra en el resultado, como en el siguiente ejemplo.

     ```
     {
         "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
         "Port": 8111,
         "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
     }
     ```

   Ahora, vuelva a ejecutar los programas, pero, esta vez, especifique el punto de enlace del clúster como parámetro en la línea de comandos.

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
    -service dax -command get-item -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

   ```
   go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
    -service dax -command query -endpoint my-cluster.l6fzcv.dax-clusters.us-
   east-1.amazonaws.com:8111
   ```

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go
 -service dax -command scan -endpoint my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com:8111
```

Fíjese en el resto del resultado y tome nota de la información sobre tiempos. Los tiempos transcurridos para las operaciones `GetItem`, `Query` y `Scan` deberían ser significativamente menores con DAX que con DynamoDB.

6. Ejecute el siguiente programa de Golang para eliminar `TryDaxGoTable`.

```
go run ~/go/pkg/mod/github.com/aws-samples/aws-dax-go-sample@v1.0.2/try_dax.go -
service dynamodb -command delete-table
```

Ejemplos de AWS SDK para Node.js 2.x

Esta sección contiene código de ejemplo para aplicaciones DAX que utilizan AWS SDK para Node.js 2.x.

Temas

- Node.js y DAX

Node.js y DAX

Siga estos pasos para ejecutar la aplicación de ejemplo de Node.js en su instancia de Amazon EC2.

Para ejecutar el ejemplo de Node.js para DAX

1. Configure Node.js en su instancia ,de Amazon EC2 de la siguiente manera:

   a. Instale el administrador de la versión de nodo (`nvm`).

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh | bash
   ```

   b. Use nvm para instalar Node.js.

   ```
   nvm install 12.16.3
   ```

   c. Compruebe que Node.js está instalado y se ejecuta correctamente.

```
node -e "console.log('Running Node.js ' + process.version)"
```

Esto debería mostrar el mensaje siguiente.

```
Running Node.js v12.16.3
```

2. Instale el cliente Node.js de DAX utilizando el administrador del paquete de nodos (npm).

```
npm install amazon-dax-client
```

3. Descargue el código fuente del programa de ejemplo (archivo `.zip`).

```
wget http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
TryDax.zip
```

Cuando haya terminado la descarga, extraiga los archivos de código fuente.

```
unzip TryDax.zip
```

4. Ejecute los siguientes programas de Node.js. El primer programa crea una tabla de Amazon DynamoDB denominada `TryDaxTable`. El segundo programa escribe datos en la tabla.

```
node 01-create-table.js
node 02-write-data.js
```

5. Ejecute los siguientes programas de Node.js.

```
node 03-getitem-test.js
node 04-query-test.js
node 05-scan-test.js
```

Tome nota de la información de tiempo; es decir, del número de milisegundos necesarios para realizar las pruebas de `GetItem`, `Query` y `Scan`.

6. En el paso anterior, ha ejecutado los programas en el punto de enlace de DynamoDB. Ejecute los programas de nuevo, pero, esta vez, las operaciones `GetItem`, `Query` y `Scan` se procesan en el clúster de DAX.

Para determinar el punto de enlace del clúster de DAX, elija una de las siguientes opciones:

- En la consola de DynamoDB: elija su clúster de DAX. El punto de enlace del clúster se muestra en la consola, como en el siguiente ejemplo.

```
dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

- En la AWS CLI: ingrese el siguiente comando.

```
aws dax describe-clusters --query "Clusters[*].ClusterDiscoveryEndpoint"
```

  El punto de enlace del clúster se muestra en el resultado, como en el siguiente ejemplo.

```
{
    "Address": "my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com",
    "Port": 8111,
    "URL": "dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com"
}
```

Ahora, vuelva a ejecutar los programas, pero, esta vez, especifique el punto de enlace del clúster como parámetro en la línea de comandos.

```
node 03-getitem-test.js dax://my-cluster.l6fzcv.dax-clusters.us-
east-1.amazonaws.com
node 04-query-test.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
node 05-scan-test.js dax://my-cluster.l6fzcv.dax-clusters.us-east-1.amazonaws.com
```

Fíjese en el resto del resultado y tome nota de la información sobre tiempos. Los tiempos transcurridos para las operaciones `GetItem`, `Query` y `Scan` deberían ser significativamente menores con DAX que con DynamoDB.

7. Ejecute el siguiente programa de Node.js para eliminar `TryDaxTable`.

```
node 06-delete-table
```

Para obtener más información sobre estos programas, consulte las siguientes secciones:

- 01-create-table.js
- 02-write-data.js

- 03-getitem-test.js
- 04-query-test.js
- 05-scan-test.js
- 06-delete-table.js

01-create-table.js

El programa `01-create-table.js` crea una tabla (`TryDaxTable`). Los demás programas de Node.js en esta sección dependerán de esta tabla.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var dynamodb = new AWS.DynamoDB(); //low-level client

var tableName = "TryDaxTable";

var params = {
  TableName: tableName,
  KeySchema: [
    { AttributeName: "pk", KeyType: "HASH" }, //Partition key
    { AttributeName: "sk", KeyType: "RANGE" }, //Sort key
  ],
  AttributeDefinitions: [
    { AttributeName: "pk", AttributeType: "N" },
    { AttributeName: "sk", AttributeType: "N" },
  ],
  ProvisionedThroughput: {
    ReadCapacityUnits: 10,
    WriteCapacityUnits: 10,
  },
};

dynamodb.createTable(params, function (err, data) {
  if (err) {
```

```
    console.error(
      "Unable to create table. Error JSON:",
      JSON.stringify(err, null, 2)
    );
  } else {
    console.log(
      "Created table. Table description JSON:",
      JSON.stringify(data, null, 2)
    );
  }
});
```

02-write-data.js

El programa `02-write-data.js` escribe datos de prueba en `TryDaxTable`.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();

var tableName = "TryDaxTable";

var someData = "X".repeat(1000);
var pkmax = 10;
var skmax = 10;

for (var ipk = 1; ipk <= pkmax; ipk++) {
  for (var isk = 1; isk <= skmax; isk++) {
    var params = {
      TableName: tableName,
      Item: {
        pk: ipk,
        sk: isk,
        someData: someData,
      },
```

```
    };

    //
    //put item

    ddbClient.put(params, function (err, data) {
      if (err) {
        console.error("Unable to write data: ", JSON.stringify(err, null, 2));
      } else {
        console.log("PutItem succeeded");
      }
    });
  }
}
```

03-getitem-test.js

El programa `03-getitem-test.js` escribe realiza operaciones `GetItem` en `TryDaxTable`.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";
```

```
var pk = 1;
var sk = 10;
var iterations = 5;

for (var i = 0; i < iterations; i++) {
  var startTime = new Date().getTime();

  for (var ipk = 1; ipk <= pk; ipk++) {
    for (var isk = 1; isk <= sk; isk++) {
      var params = {
        TableName: tableName,
        Key: {
          pk: ipk,
          sk: isk,
        },
      };

      client.get(params, function (err, data) {
        if (err) {
          console.error(
            "Unable to read item. Error JSON:",
            JSON.stringify(err, null, 2)
          );
        } else {
          // GetItem succeeded
        }
      });
    }
  }

  var endTime = new Date().getTime();
  console.log(
    "\tTotal time: ",
    endTime - startTime,
    "ms - Avg time: ",
    (endTime - startTime) / iterations,
    "ms"
  );
}
```

04-query-test.js

El programa `04-query-test.js` escribe realiza operaciones `Query` en `TryDaxTable`.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var pk = 5;
var sk1 = 2;
var sk2 = 9;
var iterations = 5;

var params = {
  TableName: tableName,
  KeyConditionExpression: "pk = :pkval and sk between :skval1 and :skval2",
  ExpressionAttributeValues: {
    ":pkval": pk,
    ":skval1": sk1,
    ":skval2": sk2,
  },
};

for (var i = 0; i < iterations; i++) {
  var startTime = new Date().getTime();
```

```
  client.query(params, function (err, data) {
    if (err) {
      console.error(
        "Unable to read item. Error JSON:",
        JSON.stringify(err, null, 2)
      );
    } else {
      // Query succeeded
    }
  });

  var endTime = new Date().getTime();
  console.log(
    "\tTotal time: ",
    endTime - startTime,
    "ms - Avg time: ",
    (endTime - startTime) / iterations,
    "ms"
  );
}
```

05-scan-test.js

El programa `05-scan-test.js` escribe realiza operaciones `Scan` en `TryDaxTable`.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");

var region = "us-west-2";

AWS.config.update({
  region: region,
});

var ddbClient = new AWS.DynamoDB.DocumentClient();
var daxClient = null;

if (process.argv.length > 2) {
  var dax = new AmazonDaxClient({
    endpoints: [process.argv[2]],
    region: region,
```

```
  });
  daxClient = new AWS.DynamoDB.DocumentClient({ service: dax });
}

var client = daxClient != null ? daxClient : ddbClient;
var tableName = "TryDaxTable";

var iterations = 5;

var params = {
  TableName: tableName,
};
var startTime = new Date().getTime();
for (var i = 0; i < iterations; i++) {
  client.scan(params, function (err, data) {
    if (err) {
      console.error(
        "Unable to read item. Error JSON:",
        JSON.stringify(err, null, 2)
      );
    } else {
      // Scan succeeded
    }
  });
}

var endTime = new Date().getTime();
console.log(
  "\tTotal time: ",
  endTime - startTime,
  "ms - Avg time: ",
  (endTime - startTime) / iterations,
  "ms"
);
```

06-delete-table.js

El programa `06-delete-table.js` elimina `TryDaxTable`. Ejecute este programa después de haber terminado las pruebas.

```
const AmazonDaxClient = require("amazon-dax-client");
var AWS = require("aws-sdk");
```

```
var region = "us-west-2";

AWS.config.update({
  region: region,
});

var dynamodb = new AWS.DynamoDB(); //low-level client

var tableName = "TryDaxTable";

var params = {
  TableName: tableName,
};

dynamodb.deleteTable(params, function (err, data) {
  if (err) {
    console.error(
      "Unable to delete table. Error JSON:",
      JSON.stringify(err, null, 2)
    );
  } else {
    console.log(
      "Deleted table. Table description JSON:",
      JSON.stringify(data, null, 2)
    );
  }
});
```

Historial de documentos de DynamoDB

En la siguiente tabla se describen los cambios importantes de cada versión de la Guía para desarrolladores de DynamoDB desde el 3 de julio de 2018 en adelante. Para recibir notificaciones sobre las actualizaciones de esta documentación, suscríbase a la fuente RSS (en la esquina superior izquierda de esta página).

Cambio	Descripción	Fecha
DAX ahora es compatible con AWS PrivateLink en las regiones comerciales y de China	Se ha agregado una guía para usar AWS PrivateLink con DynamoDB Accelerator (DAX) para ayudar a los usuarios a establecer una conectividad privada y segura con los puntos de conexión del plano de control de DAX dentro de la misma Región de AWS. Para obtener más información, consulte Generar código de infraestructura para Amazon DynamoDB mediante Consola a código.	30 de octubre de 2025
DynamoDB ahora admite el protocolo de Internet versión 6 (IPv6)	DynamoDB ahora admite las direcciones del protocolo de Internet versión 6 (IPv6) en Amazon VPC cuando se conecta a tablas, transmisiones y DAX de DynamoDB, incluso con puntos de conexión de interfaz y puerta de enlace. de AWS PrivateLink. Puede usar IPv6 para simplificar la red y cumplir con los requisitos de conformidad	3 de octubre de 2025

	con una red que admita IPv4 e IPv6.	
Se ha agregado documentación para la nueva característica de Consola a código para DynamoDB	Consola a código transforma los pasos manuales de creación de tablas de DynamoDB en la Consola de administración de AWS en un código de infraestructura automatizado y reproducible en varios formatos, como AWS CDK y CloudFormation. Para obtener más información, consulte Generar código de infraestructura para Amazon DynamoDB mediante Consola a código.	23 de septiembre de 2025
Se ha publicado la versión 3.1.0 de DynamoDB Local	Esta actualización mejora el rendimiento de las consultas PartiQL, incluida la dependencia de Joda-Time.	14 de septiembre de 2025

DynamoDB ha mejorado sus capacidades de diagnóstico de limitación con un nuevo modo de Información·de·colaboradores·de·CloudWatch, ha agregado detalles sobre las excepciones de limitación y métricas específicas de CloudWatch	DynamoDB ofrece un diagnóstico de limitación mejorado con un nuevo modo de solo `Throttled keys` en Información de colaboradores de CloudWatch que rastrea específicamente las claves de partición más limitadas , excepciones de limitación más detalladas que incluyen campos de `ThrottlingReason` estructurados, proporciona códigos de motivo y ARN de recursos específicos para ayudar a identificar exactamente qué recurso está limitado y por qué, y 16 nuevas métricas específicas de CloudWatch que corresponden a diferentes escenarios de limitación, para una supervisión precisa de eventos de limitación específicos. Para obtener más información, consulte Solución de problemas de limitación en DynamoDB.	14 de agosto de 2025

Se ha aumentado el límite para cambiar entre los modos de capacidad	Se ha aumentado el límite para cambiar una tabla del modo de capacidad aprovisionada al modo de capacidad bajo demanda de una vez cada 24 horas a cuatro veces cada 24 horas. Este cambio proporciona más flexibilidad para las cargas de trabajo que requieren cargar grandes volúmenes de datos varias veces al día o que deben adaptarse a aumentos de tráfico inesperados. Para obtener más información, consulte Aspectos a tener en cuenta al cambiar los modos de capacidad en DynamoDB.	12 de agosto de 2025
Se ha publicado la versión 3.0 de DynamoDB Local, que migra del AWS SDK para Java V1 a V2	Esta actualización elimina la dependencia de la V1, mejora el rendimiento y se alinea con la estructura de paquetes del AWS SDK V2.	17 de julio de 2025

Se ha agregado soporte para la detección optimizada de particiones en flujos de DynamoDB	El nuevo mecanismo de detección de particiones reduce el tiempo de detección inicial, mejora el rendimiento del procesamiento en estado estable, proporciona visibilidad del estado del procesamiento y mantiene una latencia de p100 constante en las operaciones a gran escala. Para obtener más información, consulte AWS PrivateLink para Amazon DynamoDB Streams.	17 de julio de 2025
Se ha agregado soporte de coherencia alta de varias regiones (MRSC) para las tablas globales de DynamoDB	DynamoDB ahora admite coherencia alta de varias regiones (MRSC) para tablas globales con una región testigo opcional. MRSC permite lecturas altamente coherentes en todas las regiones con un objetivo de punto de recuperación (RPO) igual a cero en caso de errores regionales. Las tablas globales de MRSC se deben implementar exactamente en tres regiones, con tres réplicas o con dos réplicas y un testigo. Para obtener más información y conocer las regiones disponibles, consulte Coherencia alta de varias regiones (MRSC).	30 de junio de 2025

Adaptador de la biblioteca de clientes de Kinesis (KCL) 2.x de Amazon DynamoDB Streams	DynamoDB ahora admite un adaptador de DynamoDB Streams actualizado compatible con KCL 2.x y AWS SDK v2. Para obtener más información, consulte Migrating from KCL 1.x to KCL 3.x.	11 de junio de 2025
Se ha agregado compatibilidad para ejecutar DynamoDB local en AWS CloudShell	Ahora puede ejecutar DynamoDB local directamente en el navegador a través de la Consola de administración de AWS sin necesidad de descargar ni instalar software. Para obtener más información, consulte Implementación de DynamoDB local en el equipo.	19 de mayo de 2025
Se ha agregado compatibilidad con los tipos de instancia r7i en DAX	Se han ampliado las opciones de tipos de instancia para DAX con el fin de ofrecer más flexibilidad en cuanto a rendimiento y capacidad. Para obtener más detalles sobre los nuevos tipos de instancia r7i y las capacidades, consulte Cifrado de DAX en reposo.	29 de abril de 2025

La nueva política de acceso total con alcance reducido `AmazonDynamoDBFullAccess_v2` reemplaza `AmazonDynamoDBFullAccess` .	Desde el 25 de abril de 2025, DynamoDB ha dejado de utilizar la política `AmazonDynamoDBFullAccess` y la ha reemplazado por una política de acceso total denominada `AmazonDynamoDBFullAccess_v2` . Para obtener más información, consulte Política administrada por AWS: AmazonDynamoDBFullAccess_v2.	28 de abril de 2025
DynamoDB presenta soporte de AWS PrivateLink para flujos de DynamoDB	AWS ha agregado compatibilidad con AWS PrivateLink para DynamoDB Streams, lo que permite a los clientes invocar las API de DynamoDB Streams desde Amazon Virtual Private Cloud (VPC) mediante la conectividad de red privada. Esta característica permite un acceso simplificado a la red privada sin atravesar la Internet pública, lo que respalda los requisitos de cumplimiento y seguridad para las cargas de trabajo de DynamoDB. Para obtener más información, consulte AWS PrivateLink para Amazon DynamoDB Streams.	26 de marzo de 2025

Ya está disponible el SDK de DAX para JS 3.x y Go 2.x	El SDK de DynamoDB Accelerator (DAX) para JS 3.x ya está disponible y es compatible con el AWS SDK para Java 3.x.	17 de marzo de 2025
Versión del modo de capacidad bajo demanda de NoSQL Workbench 3.13.5 para la configuración de tabla predeterminada	Al crear una tabla con la configuración predeterminada, DynamoDB crea una tabla que usa el modo de capacidad bajo demanda en lugar del modo de capacidad aprovisio nada.	24 de febrero de 2025
Nueva práctica recomendada para configurar el cliente de DAX	Publicó un nuevo tema sobre las prácticas recomenda das de la guía prescriptiva de DAX para configurar el cliente de DAX. Para obtener más información, consulte Configuración del cliente de DAX.	17 de febrero de 2025
Nuevas prácticas recomenda das para operaciones de datos masivos	Se han publicado nuevos temas de prácticas recomenda das para trabajar con modelos de datos complejos en DynamoDB. Para obtener más información, consulte Prácticas recomendadas para utilizar operaciones de datos masivos en DynamoDB.	9 de enero de 2025

Amazon DynamoDB ahora admite la recuperación en un momento dado (PITR) configurable)	DynamoDB ahora admite periodos de recuperación configurables para PITR. Ahora puede establecer el periodo PITR para cada tabla entre 1 y 35 días. Para obtener más información, consulte Copias de seguridad a un momento dado para DynamoDB.	7 de enero de 2025
Se ha publicado un nuevo tema sobre la integración de Amazon Managed Streaming para Apache Kafka con Amazon DynamoDB	Obtenga información sobre cómo Amazon Managed Streaming para Apache Kafka se integra con Amazon DynamoDB mediante la lectura de datos de temas de Apache Kafka y el almacenamiento en DynamoDB. Para obtener más información, consulte Integración de DynamoDB con Amazon Managed Streaming para Apache Kafka.	26 de diciembre de 2024
DynamoDB incorpora la compatibilidad con una integración sin ETL con SageMaker AI Lakehouse	DynamoDB incorpora una integración sin ETL que automatiza la extracción y la carga de datos de Amazon DynamoDB en el lago de datos de un cliente. Para obtener más información, consulte Integración sin ETL de DynamoDB con SageMaker AI Lakehouse.	3 de diciembre de 2024

Las tablas globales de DynamoDB ahora admiten consistencia alta de varias regiones de las tablas globales	Con la consistencia alta de varias regiones, puede crear aplicaciones de varias regiones de alta disponibilidad con un objetivo de punto de recuperación (RPO) cero, con lo que se consigue el máximo nivel de resiliencia. Para obtener más informaci ón, consulte Tablas globales con consistencia alta de varias regiones.	3 de diciembre de 2024
Actualización de las políticas administradas por DynamoDB	Se han agregado dos nuevos permisos a la política administr ada `AmazonDynamoDBRead OnlyAccess` : `dynamodb: GetAbacStatus` y `dynamodb:UpdateAba cStatus` . Estos permisos le permiten ver el estado de ABAC y habilitar ABAC para la Cuenta de AWS en la región actual. Para obtener más información, consulte AWS managed policy: AmazonDyn amoDBReadOnlyAccess.	18 de noviembre de 2024

DynamoDB incorpora compatibilidad con el control de acceso basado en atributos (ABAC)	ABAC es una estrategia de autorización que le permite definir permisos de acceso basados en etiquetas asociadas a usuarios, roles y recursos de AWS. ABAC utiliza condiciones basadas en etiquetas en las políticas de AWS Identity and Access Management (IAM) u otras políticas para permitir o denegar acciones específicas en las tablas o los índices cuando las etiquetas de las entidades principales de IAM coinciden con las de las tablas. Para obtener más información, consulte Uso del control de acceso basado en atributos con DynamoDB.	18 de noviembre de 2024
DynamoDB incorpora el rendimiento en caliente para las tablas bajo demanda y aprovisionadas	DynamoDB admite ahora el rendimiento en caliente. El rendimiento en caliente proporciona visibilidad sobre el número de operaciones de lectura y escritura que la tabla de DynamoDB puede admitir de forma instantánea, así como la capacidad de realizar un precalentamiento de las tablas de DynamoDB. Para obtener más información, consulte Rendimiento en caliente de DynamoDB.	13 de noviembre de 2024

Capacidad para consumir registros de Amazon DynamoDB Streams con Apache Flink	Ahora puede consumir registros de Amazon DynamoDB Streams con Apache Flink y aprovechar Amazon Managed Service para Apache Flink para crear y administrar rápidamente aplicaciones de procesami ento de flujos integrales. Para obtener más información, consulte DynamoDB Streams y Apache Flink.	12 de noviembre de 2024
Se han publicado nuevos temas de facturación para tablas globales y copias de seguridad	Se han publicado nuevos temas relacionados con la facturación para tablas globales y copias de seguridad . Para obtener más informaci ón, consulte Understanding Amazon DynamoDB billing for global tables y Understanding Amazon DynamoDB billing for backups.	16 de octubre de 2024
Integración sin ETL de Amazon DynamoDB con Amazon Redshift	La integración sin ETL de Amazon DynamoDB con Amazon Redshift proporcio na una canalización de ETL totalmente administrada y sin código con replicación de DynamoDB a Amazon Redshift. Para obtener más información, consulte Amazon DynamoDB zero-ETL integrati on with Amazon Redshift.	15 de octubre de 2024

Actualización solo de la documentación para agregar un tema sobre el uso de la IA generativa con DynamoDB	Se ha publicado un nuevo tema que proporciona información sobre el uso de la IA generativa con DynamoDB y que incluye ejemplos de casos de uso de la IA generativa para DynamoDB. Para obtener más información, consulte Uso de la IA generativa con DynamoDB.	11 de octubre de 2024
El SDK ahora admite puntos de conexión basados en cuentas de AWS	Se ha agregado documentación para los puntos de conexión basados en cuentas y la configuración de `ACCOUNT_ID_ENDPOINT_MODE` para los clientes del SDK. Para obtener más información, consulte la compatibilidad del SDK con puntos de conexión basados en cuentas de AWS.	3 de septiembre de 2024
Se ha rediseñado la experiencia de introducción	Se ha rediseñado la experiencia de introducción para consolidar la información y poder empezar con una incorporación más rápida. Para obtener más información, consulte Introducción a DynamoDB.	1 de agosto de 2024
Expansión del DAX a nuevas regiones de España y Suecia	DAX ya está disponible en regiones de España y Suecia. Para obtener más información, consulte Componentes del clúster de DAX.	30 de julio de 2024

Se ha reestructurado y consolidado la documentación de copia de seguridad y restauración de DynamoDB	La Guía para desarrolladores de DynamoDB tiene una nueva estructura para realizar copias de seguridad y restaurar. Para obtener más información, consulte Copia de seguridad y restauración para DynamoDB.	2 de julio de 2024
Reescritura del tema ¿Qué es Amazon DynamoDB	Se ha publicado una versión revisada y actualizada del tema ¿Qué es Amazon DynamoDB? Para obtener más información, consulte ¿Qué es Amazon DynamoDB?	21 de junio de 2024
Integración de DynamoDB Streams con EventBridge	Se ha publicado un nuevo tema sobre la integración de DynamoDB Streams con EventBridge. Para obtener más información, consulte Integrating with EventBridge.	21 de junio de 2024
Recomendaciones de DAX	Se ha publicado un nuevo tema de prácticas recomendadas que proporciona información exhaustiva sobre cómo utilizar DynamoDB Accelerator de forma eficaz. En este tema, se abordan la optimización del rendimiento, la administración de costos y las prácticas recomendadas operativas. Para obtener más información, consulte DAX prescriptive guidance.	3 de junio de 2024

Migración de una tabla de DynamoDB de una cuenta a otra	Se ha agregado un nuevo tema sobre la migración de tablas de DynamoDB de una cuenta a otra. Para obtener más información, consulte Migrating a DynamoDB table from one account to another.	29 de mayo de 2024
Se ha reestructurado y consolidado la documentación de monitoreo y registro de DynamoDB	La nueva estructura de monitoreo y registro en DynamoDB incluye tres capítulos concisos para las métricas, las operaciones de registro e Información de colaboradores.	3 de mayo de 2024

Se ha reestructurado y consolidado la documentación sobre los modos de capacidad de DynamoDB	La guía de DynamoDB ahora incluye un nuevo capítulo que contiene toda la informaci ón sobre los modos de capacidad de DynamoDB: bajo demanda y aprovisio nado. Con esta actualización, el tema Consideraciones sobre el cambio del modo de capacidad de lectura/ escritura se ha trasladad o al capítulo de prácticas recomendadas. Este tema pasa a denominarse Aspectos a tener en cuenta al cambiar los modos de capacidad e incluye información detallada sobre las prácticas recomenda das a la hora de cambiar de un modo de capacidad a otro. Además, la guía ahora incluye un nuevo capítulo con toda la información sobre las lecturas y escrituras de DynamoDB y el consumo de unidades de capacidad para las operaciones de lectura y escritura. Para obtener más información, consulte Capacidad de rendimiento de DynamoDB, Aspectos a tener en cuenta al cambiar los modos de capacidad y Lecturas y escrituras de DynamoDB.	1 de mayo de 2024

Número máximo de solicitudes bajo demanda	Ahora puede especificar el número máximo de solicitud es bajo demanda que puede realizar una tabla individual, un índice o ambos. Especific ar el rendimiento máximo bajo demanda le ayudará a mantener el uso y los costos de las tablas dentro de los límites y lo protegerá contra un aumento no intencionado de recursos consumidos. Para obtener más información, consulte Rendimiento máximo de las tablas bajo demanda.	1 de mayo de 2024
Mejoras en el generador de operaciones de NoSQL Workbench	NoSQL Workbench ahora incluye compatibilidad nativa para el modo oscuro. Se han mejorado las operaciones de tablas y elementos en el generador de operacion es. Los resultados de los elementos y la información de solicitud del generador de operaciones están disponibl es en formato JSON. Para obtener más información, consulte Generador de operaciones de NoSQL Workbench.	24 de abril de 2024

Políticas basadas en recursos para recursos de Amazon DynamoDB	Ahora DynamoDB admite las políticas basadas en recursos para tablas, índices y secuencias. Las políticas basadas en recursos permiten definir los permisos de acceso al especificar quién tiene acceso a cada recurso y las acciones que puede realizar en cada recurso. Para obtener más información, consulte Uso de políticas basadas en recursos para DynamoDB.	20 de marzo de 2024
Actualización de las políticas administradas por DynamoDB	Se ha añadido un nuevo permiso `dynamodb:GetResourcePolicy` a la política administrada `AmazonDynamoDBReadOnlyAccess` . Este permiso proporciona acceso a políticas basadas en recursos de lectura asociadas a los recursos de DynamoDB. Para obtener más información, consulte AWS managed policy: AmazonDynamoDBReadOnlyAccess.	20 de marzo de 2024

AWS PrivateLink para Amazon DynamoDB	Amazon DynamoDB ahora admite AWS PrivateLink. AWS PrivateLink le permite simplificar la conectividad de red privada entre nubes privadas virtuales (VPC), DynamoDB y sus centros de datos en las instalaciones mediante puntos de conexión de VPC de interfaz y direcciones IP privadas. Para obtener más información, consulte AWS PrivateLink para DynamoDB.	19 de marzo de 2024
Guía de programación con JavaScript	Amazon DynamoDB presenta una guía de programación para AWS SDK para JavaScript. Obtenga información sobre el AWS SDK para JavaScript, las capas de abstracción, la configuración de la conexión, la gestión de errores, la definición de las políticas de reintentos, la gestión del mantenimiento activo y mucho más. Para obtener más información, consulte Programación con JavaScript.	6 de marzo de 2024

Guía de programación con AWS SDK for Java 2.x	Se ha creado una nueva guía de programación que entra en detalle en las interfaces de documentos, de alto nivel y bajo nivel, los clientes HTTP y su configuración, la gestión de errores y que aborda los ajustes de configuración más comunes que debe tener en cuenta al usar el SDK para Java 2.x. Para obtener más información, consulte Programación de Amazon DynamoDB con AWS SDK for Java 2.x.	5 de marzo de 2024
Clonación de tablas con NoSQL Workbench	Permita a los desarroll adores utilizar NoSQL Workbench para copiar o clonar tablas entre entornos de desarrollo y regiones (DynamoDB en las instalaci ones y DynamoDB web). Para obtener más información, consulte Clonación de tablas con NoSQL Workbench.	26 de febrero de 2024

Guía de programación con Python	Se ha creado una nueva guía que entra en detalle en las bibliotecas de alto y bajo nivel y aborda los ajustes de configuración más comunes que se deben tener en cuenta al usar el SDK de Python. Para obtener más información, consulte Programación con Phyton.	5 de enero de 2024
Reescritura del tema Tiempo de vida (TTL	Se ha reescrito por completo la sección TTL de la guía. La nueva guía le ayuda a empezar a usar la caracterí stica TTL, ya que incluye fragmentos de código listos para usar. Los fragmentos de código actuales proporcio nados están en Python y Javascript. Para obtener más información, consulte TTL.	20 de diciembre de 2023
Prácticas recomendadas para interpretar los informes de facturación y uso de AWS	Se ha añadido una nueva sección donde se explican con precisión los distintos tipos de uso y los cargos correspon dientes a esos tipos de uso en DynamoDB. Para obtener más información, consulte Informes de facturación y uso.	15 de diciembre de 2023

Integración sin ETL de Amazon DynamoDB con Amazon OpenSearch Service	Amazon DynamoDB ahora admite la integración sin ETL con Amazon OpenSearch Service, que le permite realizar una búsqueda en sus datos de DynamoDB replicándolos y transformándolos automáticamente sin que se necesite código o infraestructura personalizados. Para obtener más información, consulte Integración sin ETL de DynamoDB con Amazon OpenSearch Service.	28 de noviembre de 2023
Migración desde una base de datos relacional a DynamoDB	Se ha creado una guía de migración para ayudar a los usuarios a comprender cómo se migra a DynamoDB desde una base de datos relacional.	27 de noviembre de 2023
Generar datos de muestra con NoSQL Workbench	NoSQL Workbench para Amazon DynamoDB ahora permite crear modelos de datos directamente a partir de plantillas de modelos de datos de muestra para ayudarle a diseñar esquemas de datos para sus cargas de trabajo. Puede utilizar esta característica para familiarizarse con las prácticas recomendadas de modelado de datos NoSQL al crear sus aplicaciones en DynamoDB.	28 de septiembre de 2023

Exportación incremental a S3	Ahora puede exportar los datos insertados, actualizados o eliminados, en pequeños incrementos. Con la exportación incremental, puede exportar datos modificados que van desde unos pocos megabytes hasta terabytes con unos pocos clics en la consola de administración de AWS, una llamada a la API o la interfaz de línea de comandos de AWS.	26 de septiembre de 2023
Modelado de datos para DynamoDB	Ahora puede obtener más información sobre el modelado de datos con ejemplos de DynamoDB que se centran en casos de uso específicos, sus patrones de acceso y una guía paso a paso para realizar dichos patrones de acceso.	14 de julio de 2023
Sección Solución de problemas	Ahora puede encontrar contenido de solución de problemas de latencia y limitación que puedan producirse en sus tablas de DynamoDB.	13 de marzo de 2023

Protección contra eliminación para Amazon DynamoDB	La protección contra eliminación ya está disponible para las tablas de Amazon DynamoDB en todas las regiones de AWS. DynamoDB ahora le permite proteger sus tablas de la eliminación accidental al realizar operaciones habituales de administración de tablas.	8 de marzo de 2023
CloudFormation Compatibilidad de con KDSD en tablas globales	Amazon Kinesis Data Streams para DynamoDB ahora admite CloudFormation para las tablas globales de DynamoDB, lo que significa que puede habilitar el streaming a Amazon Kinesis Data Streams en sus tablas globales de DynamoDB con las plantillas de CloudFormation.	15 de febrero de 2023
DynamoDB local admite 100 acciones por transacción	Ahora puede realizar hasta 100 acciones en una sola transacción en DynamoDB local.	9 de febrero de 2023
Uso del enfoque Well-Architected de DynamoDB para optimizar su carga de trabajo de DynamoDB	Ahora puede utilizar el enfoque Well-Architected de DynamoDB, una colección de principios de diseño y orientación que puede usar para la realización de cargas de trabajo de DynamoDB bien diseñadas.	3 de febrero de 2023

Disponibilidad de GovGloud de PartiQL	PartiQL, un lenguaje de consulta compatible con SQL para Amazon DynamoDB, es ahora compatible con AWS GovCloud (EE. UU. Este) y AWS GovCloud (EE. UU. Oeste).	21 de diciembre de 2022
Conjunto de instalación única para NoSQL Workbench y DynamoDB local	NoSQL Workbench para DynamoDB incluye ahora un proceso guiado de instalación de DynamoDB local para agilizar la configuración de su entorno de desarrollo local de DynamoDB.	6 de diciembre de 2022
Importación en bloque desde S3	Amazon DynamoDB ahora le facilita la migración y la carga de datos en nuevas tablas de DynamoDB, ya que admite la importación en bloque de datos desde Amazon S3.	18 de agosto de 2022
Integración mejorada con Service Quotas	Service Quotas ahora le permite administrar de forma proactiva las cuotas de sus cuentas y tablas. Puede ver los valores actuales, establecer alarmas para cuando la utilización de una cuota supere un umbral configurable y mucho más.	15 de junio de 2022

NoSQL Workbench agrega compatibilidad con tablas y GSI	Ahora puede utilizar NoSQL Workbench para las operaciones del plano de control de tablas e índices secundarios globales (GSI), como CreateTable, UpdateTable y DeleteTable.	2 de junio de 2022
La clase de tabla de acceso estándar poco frecuente ya está disponible en China	La clase de tabla Estándar - Acceso poco frecuente de Amazon DynamoDB ya está disponible en las regiones de China. Reduzca sus costos de DynamoDB hasta en un 60 %, mediante el uso de esta nueva clase de tabla para las tablas que almacenan datos de acceso poco frecuente.	18 de abril de 2022
Aumento de las cuotas de servicio predeterminadas y operaciones de administración de tablas	DynamoDB aumentó la cuota predeterminada del número de tablas por cuenta y región de 256 a 2500 tablas e incrementó la cantidad de operaciones simultáneas de administración de tablas de 50 a 500.	9 de marzo de 2022
Limitación opcional de elementos con PartiQL para DynamoDB	DynamoDB puede limitar el número de elementos procesados en PartiQL para las operaciones de DynamoDB como un parámetro opcional en cada solicitud.	8 de marzo de 2022

AWS Backup La integración de está disponibles en las regiones de China (Pekín y Ningxia	AWS Backup se integra ahora con DynamoDB en las regiones de China (Pekín y Ningxia). Puede satisfacer los requisitos de conformid ad y continuidad empresarial más fácilmente gracias a las características mejoradas de las copias de seguridad en AWS Backup, como las copias de seguridad entre cuentas y regiones.	26 de enero de 2022
Información de la capacidad de rendimiento mediante llamadas a la API de PartiQL	DynamoDB puede devolver la capacidad de rendimiento que consumen las llamadas a la API de PartiQL para ayudarle a optimizar sus consultas y los costos de rendimiento.	18 de enero de 2022
AWS Backup Integración de	DynamoDB ahora le ayuda a satisfacer los requisitos de conformidad y continuidad empresarial más fácilmente gracias a las características mejoradas de las copias de seguridad en AWS Backup, como las copias de seguridad entre cuentas y regiones.	24 de noviembre de 2021
Importación y exportación de conjuntos de datos de NoSQL Workbench en CSV	NoSQL Workbench para Amazon DynamoDB le permite ahora importar y rellenar automáticamente datos de muestra para ayudarle a crear y visualizar sus modelos de datos.	11 de octubre de 2021

Filtrado y recuperación de la actividad de plano de datos de Amazon DynamoDB Streams con AWS CloudTrail	Amazon DynamoDB le ofrece ahora un control más detallado del registro de auditoría al permitirle filtrar la actividad de la API de plano de datos de Streams en AWS CloudTrail.	22 de septiembre de 2021
Consola actualizada	La consola de DynamoDB es ahora su consola predeterminada para ayudarle a administrar los datos más fácilmente, simplificar las tareas comunes y ofrecerle un acceso más rápido a los recursos y las características.	25 de agosto de 2021
Ya está disponible el SDK de DAX para Java 2.x	El SDK de DynamoDB Accelerator (DAX) para Java 2.x ya está disponible y es compatible con el SDK de AWS para Java 2.x. Podrá beneficiarse de las últimas características, incluida la E/S sin bloqueo.	29 de julio de 2021
Actualizaciones de las características de NoSQL Workbench, incluidas las operaciones del plano de control	NoSQL Workbench para Amazon DynamoDB ahora le ayuda a ejecutar operaciones frecuentes con mayor facilidad para modificar los datos de la tabla y acceder a ellos.	28 de julio de 2021

Las tablas globales de DynamoDB ya están disponibles en la región de Asia-Pacífico	Las tablas globales de DynamoDB ya están disponibles en la región de Asia-Pacífico (Osaka). Puede replicar sus tablas de DynamoDB automáticamente en las 22 regiones de AWS que elija.	28 de julio de 2021
DAX ya está disponible en China	DynamoDB Accelerator (DAX) ya está disponible en la región de China (Pekín), operado por Sinnet.	28 de julio de 2021
Cifrado en tránsito de DAX	DynamoDB Accelerator (DAX) ahora admite el cifrado en tránsito de los datos entre sus aplicaciones y los clústeres de DAX, así como entre los nodos de un clúster de DAX.	24 de julio de 2021
Integración de CloudFormation y CloudTrail	Integración con AWS CloudFormation y mejoras de seguridad con el registro de plano de datos de CloudFormation.	18 de junio de 2021
CloudFormation ahora se admite para las tablas globales	Las tablas globales de Amazon DynamoDB ahora admiten AWS CloudFormation, lo que significa que puede crear tablas globales y administrar su configuración con las plantillas de CloudFormation.	14 de mayo de 2021

Compatibilidad de Amazon DynamoDB local con Java 2.x	Ahora puede utilizar el SDK de AWS para Java 2.x con DynamoDB local, la versión descargable de Amazon DynamoDB. Con DynamoDB local, puede desarrollar y probar aplicaciones con una versión de DynamoDB que se ejecuta en su entorno de desarrollo local sin incurrir en costos adicionales.	3 de mayo de 2021
NoSQL Workbench ahora admite AWS CloudFormation	NoSQL Workbench para Amazon DynamoDB ahora admite AWS CloudFormation, para que pueda administrar y modificar los modelos de datos de DynamoDB con las plantillas de CloudFormation. Además, ahora puede configurar las opciones de capacidad de las tablas en NoSQL Workbench.	22 de abril de 2021
DynamoDB y AWS Amplify ahora ofrecen integración	AWS Amplify ahora orquesta múltiples actualizaciones de índices secundarios globales de DynamoDB en una sola implementación.	20 de abril de 2021

AWS CloudTrail registra las API del plano de datos de Amazon DynamoDB Streams	Ahora puede utilizar AWS CloudTrail para registrar la actividad de la API del plano de datos de Amazon DynamoDB Streams y supervisar e investigar los cambios en los elementos de sus tablas de DynamoDB.	20 de abril de 2021
Amazon Kinesis Data Streams para Amazon DynamoDB ahora admite AWS CloudFormation	Amazon Kinesis Data Streams para Amazon DynamoDB ahora admite AWS CloudFormation, lo que significa que puede habilitar el streaming a un flujo de datos de Amazon Kinesis en sus tablas de DynamoDB con las plantillas de CloudFormation. Al transmitir los cambios de datos de DynamoDB a un flujo de datos de Kinesis, puede crear aplicaciones de streaming avanzadas con los servicios de Amazon Kinesis.	12 de abril de 2021

Amazon Keyspaces ahora ofrece puntos de conexión conformes con FIPS	Amazon Keyspaces (for Apache Cassandra) ofrece ahora puntos de conexión conformes con los Estándares federales de procesamiento de la información (FIPS) 140-2 para ayudarle a ejecutar cargas de trabajo con un elevado nivel de regulación con mayor facilidad. FIPS 140-2 es un estándar del gobierno estadounidense y canadiense que especifica los requisitos de seguridad para los módulos criptográficos que protegen la información confidencial.	8 de abril de 2021
Instancias T3 de Amazon EC2 para DAX	DAX ahora admite instancias de Amazon EC2 tipo T3, que proporcionan una línea de base de rendimiento de CPU con la posibilidad de aumentar esa línea según se necesite.	15 de febrero de 2021
Compatibilidad con NoSQL Workbench para Amazon DynamoDB para PartiQL	Ahora puede utilizar NoSQL Workbench para DynamoDB para crear expresiones PartiQL para DynamoDB.	4 de diciembre de 2020

PartiQL para DynamoDB	Ahora puede usar PartiQL para DynamoDB, un lenguaje de consulta compatible con SQL, para interactuar con las tablas de DynamoDB y ejecutar consultas ad hoc mediante la Consola de administración de AWS, la AWS Command Line Interface y las API de DynamoDB para PartiQL.	23 de noviembre de 2020
Amazon Kinesis Data Streams para Amazon DynamoDB	Ahora puede usar Amazon Kinesis Data Streams para Amazon DynamoDB con las tablas de DynamoDB para capturar los cambios a nivel del elemento y replicarlos en un flujo de datos de Kinesis.	23 de noviembre de 2020
Exportación de tablas de DynamoDB	Ahora puede exportar sus tablas de DynamoDB a Amazon S3, lo que le permite realizar análisis y consultas complejas sobre sus datos con servicios como Athena, AWS Glue, y Lake Formation.	9 de noviembre de 2020

Compatibilidad con los valores vacíos	DynamoDB ahora admite valores vacíos para atributos String (cadena) y Binary (binario) que no sean clave en tablas de DynamoDB. La compatibilidad con los valores vacíos le brinda mayor flexibilidad para usar atributos para un conjunto más amplio de casos de uso sin tener que transformar dichos atributos antes de enviarlos a DynamoDB. En los tipo de datos de conjuntos, listas y mapas, se admiten valores String (cadena) y Binary (binarios) vacíos.	18 de mayo de 2020
Compatibilidad con NoSQL Workbench para Amazon DynamoDB para Linux	Ahora NoSQL Workbench para Amazon DynamoDB es compatible con Linux, Ubuntu, Fedora y Debian.	4 de mayo de 2020

CloudWatch Contributor Insights para DynamoDB: disponible de manera general	CloudWatch Contributor Insights para DynamoDB está disponible de manera general. CloudWatch Contributor Insights para DynamoDB es una herramienta de diagnóstico que proporciona una vista rápida de las tendencias del tráfico de su tabla de DynamoDB y le ayuda a identificar las claves a las que se accede con más frecuencia (también conocidas como claves activas).	2 de abril de 2020
Actualización de tablas globales	Ahora puede actualizar sus tablas globales de la versión 2017.11.29 a la última versión de tablas globales (2019.11.21), con unos pocos clics en la consola de DynamoDB. Al actualizar la versión de las tablas globales, puede aumentar la disponibilidad de las tablas de DynamoDB fácilmente ampliando sus tablas existentes a regiones de AWS, sin necesidad de reconstruir la tabla.	16 de marzo de 2020

NoSQL Workbench para Amazon DynamoDB: disponible de manera general	NoSQL Workbench para Amazon DynamoDB se encuentra disponible de manera general. Utilice NoSQL Workbench para diseñar, crear, consultar y administrar tablas de DynamoDB.	2 de marzo de 2020
Métricas de clúster de caché de DAX	Compatibilidad con DAX para nuevas Métricas de CloudWatch, que le permiten comprender mejor el rendimiento de su clúster de DAX.	6 de febrero de 2020
CloudWatch Contributor Insights para DynamoDB: vista previa	CloudWatch Contributor Insights para DynamoDB es una herramienta de diagnóstico que proporciona una vista rápida de las tendencias del tráfico de su tabla de DynamoDB y le ayuda a identificar las claves a las que se accede con más frecuencia (también conocidas como claves activas).	26 de noviembre de 2019

Soporte de capacidad adaptable para cargas de trabajo desequilibradas	La capacidad adaptativa de Amazon DynamoDB ahora gestiona cargas de trabajo desequilibradas mejor al aislar automáticamente los elementos a los que se accede con frecuencia. Si su aplicación dirige un tráfico alto hacia uno o dos elementos de forma desproporcionada, DynamoDB volverá a equilibrar sus particiones de manera que los elementos con acceso frecuente no residan en la misma partición.	26 de noviembre de 2019
Compatibilidad con las claves administradas por el cliente	DynamoDB ahora admite claves administradas por el cliente, lo que significa que puede tener control total sobre cómo cifrar y administrar la seguridad de sus datos de DynamoDB.	25 de noviembre de 2019
Compatibilidad con NoSQL Workbench para DynamoDB local (versión descargable)	NoSQL Workbench ahora admite conectarse a DynamoDB local (versión descargable) para diseñar, crear, consultar y administrar tablas de DynamoDB.	8 de noviembre de 2019

NoSQL Workbench: vista previa	Es la versión inicial de NoSQL Workbench para DynamoDB. Utilice NoSQL Workbench para diseñar, crear, consultar y administrar tablas DynamoDB. Para obtener más información, consulte NoSQL Workbench para Amazon DynamoDB (vista previa).	16 de septiembre de 2019
DAX incorpora compatibilidad con operaciones transaccionales con Python y .NET	DAX admite las API `TransactWriteItems` y `TransactGetItems` para aplicaciones escritas en Go, Java, .NET, Node.js y Python. Para obtener más información, consulte Acceleration en memoria con DAX.	14 de febrero de 2019
Actualizaciones de Amazon DynamoDB local (versión descargable	DynamoDB local (versión descargable) admite ahora API transaccionales, capacidad de lectura/escritura bajo demanda, informes de capacidad para las operaciones de lectura y escritura, y 20 índices secundarios globales. Para obtener más información, consulte Diferencias entre la versión descargable de DynamoDB y el servicio web de DynamoDB.	4 de febrero de 2019

Amazon DynamoDB bajo demanda	DynamoDB bajo demanda es una opción de facturación flexible que permite atender a miles de solicitudes por segundo sin tener que planificar la capacidad. DynamoDB bajo demanda ofrece precios de pago por solicitud para las solicitudes de lectura y escritura. De este modo, solo tendrá que pagar por aquello que utilice. Para obtener más información, consulte Capacidad de rendimiento de DynamoDB.	28 de noviembre de 2018
Amazon DynamoDB Transactions	Las transacciones de DynamoDB realizan cambios coordinados de tipo "todo o nada" en varios elementos de la misma o de distintas tablas. De este modo, aportan atomicidad, consistencia, aislamiento y durabilidad (ACID) en DynamoDB. Para obtener más información, consulte Amazon DynamoDB Transactions.	27 de noviembre de 2018

Amazon DynamoDB cifra todos los datos en reposo del cliente	El cifrado en reposo de DynamoDB proporciona una capa adicional de seguridad de los datos, porque los protege en una tabla cifrada que incluye su clave principal, los índices secundarios locales y globales, las secuencias, las tablas globales, las copias de seguridad y los clústeres DAX siempre que los datos se almacenen en un soporte duradero. Para obtener más información, consulte Cifrado en reposo de Amazon DynamoDB.	15 de noviembre de 2018
Uso más sencillo de Amazon DynamoDB Local con la nueva imagen de Docker	Ya es más sencillo utilizar DynamoDB local, la versión descargable de DynamoDB, para ayudarle a desarrollar y probar sus aplicaciones de DynamoDB usando la nueva imagen de Docker de DynamoDB local. Para obtener más información, consulte DynamoDB (versión descargable) y Docker.	22 de agosto de 2018

DynamoDB Accelerator (DAX) incorpora la compatibilidad con el cifrado en reposo	Ahora, DynamoDB Accelerator (DAX) admite el cifrado en reposo para los nuevos clústeres de DAX. De este modo, le ayuda a acelerar las lecturas de las tablas de Amazon DynamoDB en aplicaciones sensibles a la seguridad que están sujetas a estrictos requisitos normativos y de cumplimiento. Para obtener más información, consulte Cifrado en reposo de DAX.	9 de agosto de 2018
La recuperación a un momento dado (PITR) de DynamoDB agrega compatibilidad para restaurar tablas eliminadas	Si elimina una tabla con la característica de recuperación a un momento dado habilitada, se crea un backup del sistema y se mantiene durante 35 días (sin costo adicional). Para obtener más información, consulte Antes de empezar a usar la recuperación a un momento dado.	7 de agosto de 2018
Actualizaciones ahora disponibles sobre RSS	Ahora puede suscribirse a la fuente RSS (en la esquina superior izquierda de esta página) para recibir notificaciones sobre actualizaciones de la Guía para desarrolladores de Amazon DynamoDB.	3 de julio de 2018

Actualizaciones anteriores

En la siguiente tabla se describen los cambios importantes de la Guía para desarrolladores de DynamoDB antes del 3 de julio de 2018.

Cambio	Descripción	Fecha de modificación
Ir a compatibilidad con DAX	Ahora, puede habilitar el rendimiento de lectura en microsegundos para las tablas Amazon DynamoDB en sus aplicaciones escritas en el lenguaje de programación Go utilizando el nuevo SDK para Go de DynamoDB Accelerat or(DAX). Para obtener más información, consulte SDK para Go de DAX.	26 de junio de 2018
DynamoDB anuncia SLA	DynamoDB ha lanzado un SLA de disponibilidad pública. Para obtener más informaci ón, consulte el Acuerdo de nivel de servicios de Amazon DynamoDB.	19 de junio de 2018
Backups continuos de DynamoDB y recuperación a un momento dado (PITR)	La recuperación a un momento dado ayuda a proteger las tablas de Amazon DynamoDB de operaciones accidentales de escritura o eliminación. Al habilitar la recuperación a un momento dado, ya no hay que preocuparse por crear, mantener o planifica r backups bajo demanda.	25 de abril de 2018

Cambio	Descripción	Fecha de modificación
	Por ejemplo, imaginemos que un script de prueba escribe accidentalmente en una tabla de DynamoDB. Con la recuperación a un momento dado, puede restaurar la tabla a cualquier momento de los últimos 35 días. DynamoDB mantiene backups acumulativos de la tabla. Para obtener más información, consulte Copias de seguridad a un momento dado para DynamoDB.	
Cifrado en reposo de DynamoDB	El cifrado en reposo de DynamoDB, disponible para nuevas tablas de DynamoDB, ayuda a proteger los datos de aplicaciones en las tablas de Amazon DynamoDB utilizando las claves de cifrado administradas por AWS almacenadas en AWS Key Management Service. Para obtener más información, consulte Cifrado en reposo en DynamoDB.	8 de febrero de 2018

Cambio	Descripción	Fecha de modificación
Backup y restauración de DynamoDB	La función de backup bajo demanda permite crear backups completos de los datos de las tablas de DynamoDB para archivarlos, lo que le ayuda a satisfacer los requisitos reglamentarios corporativos y gubername ntales. Puede crear backups de las tablas que ocupen desde unos pocos megabytes a cientos de terabytes de datos, sin impacto alguno en el rendimiento y la disponibi lidad de las aplicaciones de producción. Para obtener más información, consulte Copia de seguridad y restauración para DynamoDB.	29 de noviembre de 2017

Cambio	Descripción	Fecha de modificación
Tablas globales de DynamoDB	Las tablas globales se crean en la huella global de DynamoDB para proporcionarle una base de datos completamente administrada, multi regional y multi activa, que proporciona un rendimiento rápido de las operaciones locales de lectura y escritura para aplicaciones globales a escala masiva. Las tablas globales replican automáticamente las tablas de Amazon DynamoDB en las regiones de AWS de su elección. Para obtener más información, consulte Tablas globales: multiactivas, replicación en varias regiones.	29 de noviembre de 2017
Compatibilidad de Node.js para DAX	Los desarrolladores de Node.js pueden aprovechar DynamoDB Accelerator (DAX), utilizando el cliente de DAX para Node.js. Para obtener más información, consulte Aceleración en memoria con DynamoDB Accelerator (DAX).	5 de octubre de 2017

Cambio	Descripción	Fecha de modificación
Puntos de enlace de la VPC para DynamoDB	Los puntos de enlace de DynamoDB permiten que las instancias de Amazon EC2 de su Amazon VPC accedan a DynamoDB, sin exponerse a la infraestructura pública de Internet. El tráfico de red entre su VPC y DynamoDB no sale de la red de Amazon. Para obtener más información, consulte Uso de puntos de conexión de Amazon VPC para tener acceso a DynamoDB.	16 de agosto de 2017

Cambio	Descripción	Fecha de modificación
Auto Scaling para DynamoDB	El escalado automático de DynamoDB elimina la necesidad de definir o ajustar manualmente los ajustes de rendimiento aprovisionado. En su lugar, el escalado automático de DynamoDB ajusta dinámicamente la capacidad de lectura y escritura para responder a los patrones de tráfico reales. Esto permite a una tabla o índice secundari o global incrementar su capacidad de lectura y escritura aprovisionada para abastecer increment os repentinos del tráfico sin limitaciones. Cuando la carga de trabajo disminuye , el escalado automático de DynamoDB reduce la capacidad aprovisionada. Para obtener más informaci ón, consulte Administración automática de la capacidad de rendimiento con la función Auto Scaling de DynamoDB.	14 de junio de 2017

Cambio	Descripción	Fecha de modificación
DynamoDB Accelerator (DAX)	DynamoDB Accelerator (DAX) es una caché en memoria altamente disponible y completamente administrada para DynamoDB que multiplica el rendimiento hasta por diez (de milisegundos a microsegundos) incluso con millones de solicitudes por segundo. Para obtener más información, consulte Aceleración en memoria con DynamoDB Accelerator (DAX).	19 de abril de 2017
DynamoDB admite ahora el vencimiento automático de elementos con período de vida (TTL)	La función de período de vida (TTL) de Amazon DynamoDB lo habilita a eliminar automáticamente los elementos de las tablas que han vencido, sin ningún coste adicional. Para obtener más información, consulte Uso del período de vida (TTL) en DynamoDB.	27 de febrero de 2017
DynamoDB admite ahora el etiquetado de asignación de coste	A partir de ahora, puede agregar etiquetas a las tablas de Amazon DynamoDB para mejorar el proceso de categorización del uso y generar informes de costos más granulares. Para obtener más información, consulte Cómo agregar etiquetas a los recursos en DynamoDB.	19 de enero de 2017

Cambio	Descripción	Fecha de modificación
Nueva API `DescribeLimits` de DynamoDB	La API `DescribeLimits` devuelve los límites de capacidad aprovisionada para su cuenta de AWS en una región, tanto para la región en su conjunto como para cualquier tabla de DynamoDB que se cree en ella. La API permite determinar cuáles son los límites actuales en el nivel de cuenta, para que pueda compararlos con la capacidad aprovisionada que está utilizando y disponga de tiempo suficiente para solicitar un aumento antes de agotarla. Para obtener más información, consulte Cuotas en Amazon DynamoDB y DescribeLimits en la Referencia de la API de Amazon DynamoDB.	1 de marzo de 2016

Cambio	Descripción	Fecha de modificación
Actualización de la consola de DynamoDB y nueva terminología relativa a los atributos de clave principal	La consola de administración de DynamoDB se ha rediseñado para que resulte más intuitiva y fácil de usar. Durante esta actualización, hemos introducido una terminología nueva para los atributos de clave principal: • Clave de partición, denominada también atributo hash. • Clave de ordenación, denominada también atributo de intervalo. Solo se han modificado los nombres; la funcionalidad sigue siendo la misma. Al crear una tabla o un índice secundario, puede elegir entre una clave principal simple (que solo tiene una clave de partición) o compuesta (que consta de una clave de partición y una clave de ordenación). La documentación de DynamoDB se ha actualizado para reflejar estos cambios.	12 de noviembre de 2015

Cambio	Descripción	Fecha de modificación
Amazon DynamoDB Storage Backend para Titan	El DynamoDB Storage Backend para Titan es un paquete de almacenamiento backend para la base de datos de gráficos Titan que se implementa en Amazon DynamoDB. Cuando se usa DynamoDB Storage Backend para Titan, los datos se benefician de la protección de DynamoDB, que abarca los centros de datos de alta disponibilidad de Amazon. El complemento está disponible para Titan versión 0.4.4 (principalmente con fines de compatibilidad con las aplicaciones existentes) y Titan versión 0.5.4 (recomendado para nuevas aplicaciones). Al igual que en los demás paquetes de almacenamiento backend para Titan, este complemento es compatible con el stack Tinkerpop (versiones 2.4 y 2.5), incluidos la API Blueprints y el shell Gremlin.	20 de agosto de 2015

Cambio	Descripción	Fecha de modificación
DynamoDB Streams, replicación entre regiones y examen con lecturas fuertemente consistentes	DynamoDB Streams captura una secuencia en orden cronológico de las modificaciones de los elementos en una tabla de DynamoDB y almacena esta información en un registro durante un máximo de 24 horas. Las aplicaciones pueden obtener acceso a este registro y ver los elementos de datos tal y como se encontraban antes y después de la modificación, prácticamente en tiempo real. Para obtener más información, consulte Captura de datos de cambios para DynamoDB Streams y la Referencia de la API de DynamoDB Streams. La replicación entre regiones de DynamoDB es una solución del lado del cliente para mantener copias idénticas de tablas de DynamoDB en distintas regiones de AWS, casi en tiempo real. Puede utilizar la replicación entre regiones para realizar backups de las tablas de DynamoDB u ofrecer acceso de baja latencia a los datos cuando los usuarios se encuentran distribuidos geográficamente.	16 de julio de 2015

Cambio	Descripción	Fecha de modificación
	De forma predeterminada, la operación `Scan` de DynamoDB usa las lecturas eventualmente consistentes. Si lo prefiere, puede establecer el parámetro `ConsistentRead` en true para usar lecturas de consistencia alta. Para obtener más información, consulte Coherencia de lectura para el análisis y Scan en la Referencia de la API de Amazon DynamoDB.	
AWS CloudTrailCompatibilidad con para Amazon DynamoDB	DynamoDB ya está integrada con CloudTrail. CloudTrail captura las llamadas al API realizadas desde la consola de DynamoDB o desde el API de DynamoDB y las registra en archivos de registros. Para obtener más información, consulte Registrar las operaciones de DynamoDB mediante AWS CloudTrail y la Guía del usuario de la AWS CloudTrail.	28 de mayo de 2015

Cambio	Descripción	Fecha de modificación
Compatibilidad mejorada con expresiones de consulta	En esta versión, se agrega un nuevo parámetro `KeyConditionExpression` al API `Query`. Una operación `Query` lee los elementos de una tabla o un índice usando los valores de clave principal. El parámetro `KeyConditionExpression` es una cadena que identifica los nombres de las claves principales y las condiciones que se deben aplicar a los valores de claves; `Query` recupera solamente aquellos elementos que satisfacen la expresión. La sintaxis de `KeyConditionExpression` es parecida a la de otros parámetros de expresión de DynamoDB y le permite definir variables de sustitución para los nombres y valores contenidos en la expresión. Para obtener más información, consulte Consulta de tablas en DynamoDB.	27 de abril de 2015

Cambio	Descripción	Fecha de modificación
Nuevas funciones de comparación para escrituras condicionales	En DynamoDB, el parámetro `ConditionExpression` determina si una operación `PutItem`, `UpdateItem` o `DeleteItem` se llevará a cabo correctamente: el elemento se escribe solamente si la condición se evalúa en true (verdadero). En esta versión se agregan dos funciones nuevas, `attribute_type` y `size`, para usarlas con `ConditionExpression`. Estas funciones permiten llevar a cabo escrituras condicionales según el tipo o de datos o el tamaño de un atributo de la tabla. Para obtener más información, consulte Ejemplo de la CLI de expresión de condición de DynamoDB.	27 de abril de 2015

Cambio	Descripción	Fecha de modificación
API Scan para índices secundarios	En DynamoDB, una operación `Scan` lee todos los elementos de una tabla, aplica criterios de filtrado definidos por el usuario y devuelve a la aplicación los elementos de datos seleccionados. Esta misma funcionalidad se encuentra ahora disponible para los índices secundari os. Para examinar un índice secundario local o global, se especifican los nombres del índice y de su tabla principal . De forma predeterminada, la operación `Scan` aplicada a un índice devuelve todos los datos que este contiene; sin embargo, puede utilizar una expresión de filtro para delimitar los resultados que se devuelven a la aplicación. Para obtener más información, consulte Análisis de tablas en DynamoDB.	10 de febrero de 2015

Cambio	Descripción	Fecha de modificación
Operaciones online con índices secundarios globales	La indexación online permite agregar o eliminar índices secundarios globales de las tablas existentes. Con la indexación online, no es necesario definir todos los índices de una tabla al crearla, sino que puede agregar un índice nuevo en cualquier momento. Del mismo modo, si decide que ya no necesita un índice, puede eliminarlo cuando lo desee. Las operaciones de indexación online no aplican bloqueos, de modo que la tabla permanece disponible para la actividad de lectura y escritura mientras se agregan o eliminan los índices. Para obtener más información, consulte Administración de índices secundarios globales en DynamoDB.	27 de enero de 2015

Cambio	Descripción	Fecha de modificación
Compatibilidad del modelo de documento con JSON	DynamoDB le permite almacenar y recuperar documentos con plena compatibilidad con los modelos de documentos. Los tipos de datos nuevos son totalmente compatibles con el estándar JSON y permiten anidar los componentes de documentos unos en otros. Puede utilizar los operadores de desreferenciación de rutas de documentos para leer y escribir componentes individuales, sin tener que recuperar todo el documento. En esta versión también se incorporan nuevos parámetros de expresión para especificar proyecciones, condiciones y acciones de actualización al leer o escribir elementos de datos. Para obtener más información sobre la compatibilidad de los modelos de documentos con JSON, consulte Tipos de datos y Uso de expresiones en DynamoDB.	7 de octubre de 2014

Cambio	Descripción	Fecha de modificación
Escalado flexible	Para las tablas y los índices secundarios globales, puede aumentar la capacidad de rendimiento aprovisionado de lectura y escritura en cualquier cantidad, siempre y cuando no rebase los límites de cada cuenta o tabla. Para obtener más información, consulte Cuotas en Amazon DynamoDB.	7 de octubre de 2014
Tamaños de elementos mayores	El tamaño máximo de un elemento en DynamoDB se ha aumentado de 64 a 400 KB. Para obtener más información, consulte Cuotas en Amazon DynamoDB.	7 de octubre de 2014

Cambio	Descripción	Fecha de modificación
Expresiones condicionales mejoradas	En DynamoDB se han ampliado los operadores que están disponibles para las expresiones condicionales, lo que le aporta más flexibilidad en las operaciones de colocación, actualización y eliminación condicionales. Los nuevos operadores disponibles le permiten comprobar si un atributo existe o no, si es mayor o igual que un valor determinado, si está comprendido entre dos valores o comienza por unos caracteres en concreto, entre otras condiciones. DynamoDB también proporciona un operador OR opcional para evaluar condiciones múltiples. De forma predeterminada, se utiliza AND entre las condiciones de una expresión, de tal forma que esta solo será true si lo son todas las condiciones que contiene. Si especifica OR en su lugar, la expresión será true si una o varias condiciones lo son. Para obtener más información, consulte Uso de elementos y atributos en DynamoDB.	24 de abril de 2014

Cambio	Descripción	Fecha de modificación
Filtro de consulta	La API de DynamoDB `Query` admite una nueva opción `QueryFilter` . De forma predeterminada, la operación `Query` busca los elementos que coinciden con un determinado valor de clave de partición y una condición de clave de ordenación opcional. Un filtro de `Query` aplica expresiones condicion ales a otros atributos sin clave; si hay un filtro `Query` presente, los elementos que no coinciden con las condicion es del filtro se descartan antes de devolver los resultados de `Query` a la aplicación. Para obtener más información, consulte Consulta de tablas en DynamoDB.	24 de abril de 2014

Cambio	Descripción	Fecha de modificación
Exportación e importación de datos mediante la Consola de administración de AWS	La consola de DynamoDB se ha mejorado para simplificar la exportación e importación de datos en las tablas de DynamoDB. Con tan solo unos clics, puede configurar una AWS Data Pipeline para organizar el flujo de trabajo y un clúster de Amazon Elastic MapReduce para copiar los datos de las tablas de DynamoDB a un bucket de Amazon S3, o viceversa. Puede realizar la importación o exportación una sola vez o configurar un trabajo de exportación diario. Incluso puede realizar importaciones y exportaciones entre regiones copiando los datos de DynamoDB de una tabla en una región de AWS a una tabla de otra región de AWS.	6 de marzo de 2014

Cambio	Descripción	Fecha de modificación
Reorganización de la documentación de la API de nivel superior	Ahora resulta más fácil encontrar información sobre las API siguientes: • Java: DynamoDBMappper • .NET: modelo de documento y modelo de persistencia de objetos Estas API de nivel superior se documentan aquí: Interfaces de programación de nivel superior para DynamoDB.	20 de enero de 2014

Cambio	Descripción	Fecha de modificación
Índices secundarios globales	DynamoDB agrega compatibilidad para los índices secundarios globales. Al igual que sucede con un índice secundario local, un índice secundario global se define mediante una clave alternativa de una tabla y, a continuación, se emiten solicitudes Query para consultarlo. A diferencia de un índice secundario local, la clave de partición del índice secundario global no tiene que ser la misma que la clave de partición de la tabla, sino que puede ser cualquier atributo escalar de esta última. La clave de ordenación es opcional y también puede ser cualquier atributo escalar de la tabla. Además, un índice secundario global presenta sus propios ajustes de rendimiento aprovisionado, que son independientes de los de su tabla principal. Para obtener más información, consulte Mejora del acceso con índices secundarios en DynamoDB y Uso de índices secundarios globales en DynamoDB.	12 de diciembre de 2013

Cambio	Descripción	Fecha de modificación
Control de acceso detallado	DynamoDB incorpora compatibilidad con el control de acceso preciso. Esta característica permite a los clientes especificar qué entidades principal es (usuarios, grupos o roles) pueden acceder a los elementos y atributos individuales de una tabla o un índice secundario de DynamoDB. Las aplicacio nes también pueden sacar partido de las identidades web federadas para delegar la tarea de autenticar a los usuarios en un proveedor de identidades tercero, como Facebook, Google o Login with Amazon. De esta forma, las aplicaciones (incluidas las aplicaciones para móviles) pueden administrar grandes cantidades de usuarios y, al mismo tiempo, asegurarse de que nadie pueda acceder a los elementos de datos de DynamoDB a menos que estén autorizados para ello. Para obtener más información, consulte Uso de condiciones de las políticas de IAM para control de acceso preciso.	29 de octubre de 2013

Cambio	Descripción	Fecha de modificación
Unidades de capacidad de lectura de 4 KB	El tamaño de la unidad de capacidad de lectura ha aumentado de 1 a 4 KB. Esta mejora permite reducir el número de unidades de capacidad de lectura aprovisionadas que se requieren para muchas aplicaciones. Por ejemplo, antes de esta versión, para leer un elemento de 10 KB se consumían 10 unidades de capacidad de lectura; ahora, la misma lectura de 10 KB solo consumiría 3 unidades (10 KB/4 KB, redondeados al múltiplo de 4 KB inmediatamente superior). Para obtener más información, consulte Capacidad de rendimiento de DynamoDB.	14 de mayo de 2013

Cambio	Descripción	Fecha de modificación
Exámenes en paralelo	DynamoDB agrega compatibilidad con las operaciones Scan en paralelo. Ahora, las aplicaciones pueden dividir una tabla en segmentos lógicos y examinar todos ellos simultáneamente. Esta característica reduce el tiempo necesario para que se complete una operación Scan y utiliza plenamente la capacidad de lectura aprovisionada de la tabla. Para obtener más información, consulte Análisis de tablas en DynamoDB.	14 de mayo de 2013
Índices secundarios locales	DynamoDB agrega compatibilidad para los índices secundarios locales. Se pueden definir índices de clave de ordenación basados en atributos sin clave y, a continuación, utilizarlos en solicitudes Query. El uso de uno o varios índices secundarios permite que las aplicaciones recuperen de manera eficiente elementos de datos que abarcan varias dimensiones. Para obtener más información, consulte Índices secundarios locales.	18 de abril de 2013

Cambio	Descripción	Fecha de modificación
Nueva versión de la API	En esta versión, DynamoDB introduce una nueva versión de la API (2012-08-10). La versión de la API anterior (2011-12-05) se admite todavía con fines de compatibilidad retroactiva con las aplicaciones existentes. Las nuevas aplicaciones deben utilizar la nueva versión de la API, 2012-08-10. Recomendamos migrar las aplicaciones existentes a la versión 2012-08-10 de la API, porque las nuevas características de DynamoDB (como los índices secundarios locales) no se podrán aplicar retroactivamente en la versión anterior de la API. Para obtener más información acerca de la API versión 2012-08-10, consulte la Referencia de la API de Amazon DynamoDB.	18 de abril de 2013

Cambio	Descripción	Fecha de modificación
Compatibilidad con variables de políticas de IAM	El lenguaje de la política de acceso de IAM ya es compatible con variables . Cuando se evalúa una política, las variables de la política se sustituyen por valores facilitados por información basada en contexto desde la sesión del usuario autenticado. Puede utilizar las variables de políticas para definir políticas con un propósito general sin mostrar un listado explícito con todos los componentes de la política. Para obtener más información sobre las variables de políticas, visite Variables de políticas en la guía AWS Identity and Access Management mediante IAM. Para obtener algunos ejemplos de variables de políticas de DynamoDB, consulte Identity and Access Management en Amazon DynamoDB.	4 de abril de 2013

Cambio	Descripción	Fecha de modificación
Actualización de los ejemplos de código PHP para la versión 2 de AWS SDK para PHP	Ya está disponible la versión 2 de AWS SDK para PHP Los ejemplos de código PHP de la Guía de desarrolladores de Amazon DynamoDB se han actualizado para utilizar este nuevo SDK. Para obtener más información acerca de la versión 2 del SDK , consulte AWS SDK para PHP.	23 de enero de 2013
Nuevo punto de conexión	DynamoDB se expande a la Región GovCloud (EE.UU. Oeste) de AWS. Para obtener la lista actualizada de puntos de enlace de servicio y protocolos, consulte Regiones y puntos de enlace.	3 de diciembre de 2012
Nuevo punto de conexión	DynamoDB se expande a la región de América del Sur (São Paulo). Para obtener la lista actualizada de puntos de enlace admitidos, consulte Regiones y puntos de enlace.	3 de diciembre de 2012
Nuevo punto de conexión	DynamoDB se expande a la región Asia Pacífico (Sídney). Para obtener la lista actualiza da de puntos de enlace admitidos, consulte Regiones y puntos de enlace.	13 de noviembre de 2012

Cambio	Descripción	Fecha de modificación
En DynamoDB se implement a la compatibilidad con las sumas de comprobación CRC32, se admiten las operaciones de obtención por lotes de consistencia alta y se eliminan las restricciones a las actualizaciones simultáneas de tablas.	• DynamoDB calcula una suma de comprobación CRC32 de la carga de HTTP y devuelve esta suma de comprobación en un nuevo encabezado, `x-amz-crc32` . Para obtener más información, consulte API de bajo nivel de DynamoDB. • De forma predeterminada, las operaciones de lectura realizadas mediante la API `BatchGetItem` son consistentes finales. Un nuevo parámetro `ConsistentRead` de `BatchGetItem` permite elegir lecturas de consisten cia alta en su lugar para las tablas incluidas en la solicitud. Para obtener más información, consulte Descripción. • En esta versión se eliminan algunas restricciones que se aplicaban a la actualiza ción simultánea de varias tablas. El número total de tablas que se pueden actualizar a la vez sigue siendo 10; sin embargo, ahora pueden encontrar se en cualquier combinaci ón de estados `CREATING`,	2 de noviembre de 2012

Cambio	Descripción	Fecha de modificación
	`UPDATING` o `DELETING`. Tampoco se aplica ya ninguna cantidad mínima para aumentar o reducir los valores de ReadCapacityUnits o WriteCapacityUnits de una tabla. Para obtener más información, consulte Cuotas en Amazon DynamoDB.	
Documentación de prácticas recomendadas	En la Guía de desarrollador de Amazon DynamoDB se identifican las prácticas recomendadas para usar tablas y elementos, así como recomendaciones sobre las operaciones de consulta y análisis.	28 de septiembre de 2012

Cambio	Descripción	Fecha de modificación
Compatibilidad con el tipo de datos Binary	Además de los tipos Number (número) y String (cadena), a partir de ahora DynamoDB admite el tipo de datos Binary (binario). Antes de esta versión, para almacenar datos binarios había que convertirlos en cadenas antes de almacenar los en DynamoDB. Además del trabajo de conversión en el lado del cliente, este proceso solía aumentar el tamaño del elemento de datos, con lo que se requería más espacio de almacenamiento y, posibleme nte, mayor capacidad de rendimiento aprovisionada. Ahora, los atributos de tipo Binary permiten almacenar datos binarios de cualquier índole, tales como datos comprimidos, datos cifrados o imágenes. Para obtener más información consulte Tipos de datos. Para obtener ejemplos funcionales de cómo utilizar datos de tipo binario en los SDK de AWS, consulte las siguientes secciones: • Ejemplo: control de atributos de tipo binario mediante la	21 de agosto de 2012

Cambio	Descripción	Fecha de modificación
	API de documentos de AWS SDK para Java • Ejemplo: control de atributos de tipo binario mediante la API de bajo nivel de AWS SDK para .NET Para obtener la compatibilidad adicional con el tipo de datos Binary en los SDK de AWS, deberá descargar los SDK más recientes y puede que también tenga que actualizar las aplicaciones existentes. Para obtener más información sobre la descarga de los SDK de AWS, consulte Ejemplos de código .NET.	
Los elementos de las tablas de DynamoDB se pueden actualizar y copiar desde la consola de DynamoDB	Además de agregar y eliminar elementos de tablas, ahora los usuarios de DynamoDB también pueden actualizarlos y copiarlos desde la consola de DynamoDB. Esta nueva funcionalidad simplifica la modificación de elementos individuales en la consola.	14 de agosto de 2012

Cambio	Descripción	Fecha de modificación
DynamoDB reduce los requisitos mínimos de rendimiento por tabla	Ahora, DynamoDB admite requisitos más bajos de rendimiento de cada tabla; en concreto, 1 unidad de capacidad de escritura y 1 unidad de capacidad de lectura. Para obtener más información, consulte el tema Cuotas en Amazon DynamoDB en la Guía para desarrolladores de Amazon DynamoDB.	9 de agosto de 2012
Compatibilidad con Signature Version 4	DynamoDB ya es compatible con Signature Version 4 para autenticar solicitudes.	5 de julio de 2012
Compatibilidad con el explorador de tablas en la consola de DynamoDB	La consola de DynamoDB ya es compatible con un explorador de tablas que le permite consultar y examinar los datos contenidos en ellas. También puede insertar nuevos elementos o eliminar los existentes. Las secciones `SampleData` y Uso de la consola se han actualizado para incluir estas nuevas características.	22 de mayo de 2012

Cambio	Descripción	Fecha de modificación
Nuevos puntos de enlace	La disponibilidad de DynamoDB se expande con nuevos puntos de enlace en la región EE. UU. Oeste (Norte de California), la región EE. UU. Oeste (Oregón) y la región de Asia Pacífico (Singapur). Para obtener la lista actualizada de puntos de enlace admitidos, visite Regiones y puntos de enlace.	24 de abril de 2012
Compatibilidad con la API BatchWriteItem	DynamoDB admite ahora una API de escritura en lote que permite colocar y eliminar varios elementos de una o varias tablas en una única llamada a la API. Para obtener más información acerca de la API de escritura en lote de DynamoDB, consulte BatchWriteItem. Para obtener más información sobre cómo usar elementos y aplicar la característica de escritura en lote en los SDK de AWS, consulte Uso de elementos y atributos en DynamoDB y Ejemplos de código .NET.	19 de abril de 2012

Cambio	Descripción	Fecha de modificación
Más códigos de error documentados	Para obtener más información, consulte Control de errores con DynamoDB.	5 de abril de 2012
Nuevo punto de conexión	DynamoDB se expande a la región Asia Pacífico (Tokio). Para obtener la lista actualizada de puntos de enlace admitidos, consulte Regiones y puntos de enlace.	29 de febrero de 2012
`ReturnedItemCount` Se agrega la métrica	Una nueva métrica, `ReturnedItemCount` , que proporciona el número de elementos devueltos en la respuesta a una operación de consulta o análisis para DynamoDB, está disponible para monitoreo en CloudWatch.	24 de febrero de 2012
Se agregan ejemplos sobre el incremento de valores	DynamoDB permite incrementar y disminuir los valores numéricos existentes. Se incluyen ejemplos de cómo agregar valores a otros existentes en los apartados "Actualización de un elemento" de las secciones : Uso de elementos: Java. Uso de elementos: .NET.	25 de enero de 2012

Cambio	Descripción	Fecha de modificación
Lanzamiento del producto inicial	Se presenta la versión Beta de DynamoDB como nuevo servicio.	18 de enero de 2012

Características heredadas de DynamoDB

Los siguientes temas tratan sobre características heredadas que DynamoDB todavía admite. No se ha realizado ningún desarrollo activo en estas características.

Temas

- Versión 2017.11.29 (heredada) de las tablas globales
- Versión anterior de la API de DynamoDB de bajo nivel (2011-12-05)
- Parámetros condicionales de DynamoDB heredados

Versión 2017.11.29 (heredada) de las tablas globales

> **Important**
>
> Esta documentación corresponde a la versión 2017.11.29 (heredada) de las tablas globales, lo que debe evitarse en el caso de las tablas globales nuevas. Los clientes deberían utilizar la versión 2019.11.21 (actual) de las tablas globales siempre que sea posible, ya que proporciona mayor flexibilidad, mayor eficacia y consume menos capacidad de escritura que la 2017.11.29 (heredada).
> Para determinar qué versión utiliza, consulte Determinación de la versión de una tabla global.
> Para actualizar las tablas globales existentes de la versión 2017.11.29 (heredada) a la versión 2019.11.21 (actual), consulte Versiones de tablas globales de DynamoDB.

Temas

- Tablas globales: cómo funcionan
- Prácticas recomendadas y requisitos para la administración de tablas globales
- Creación de una tabla global
- Monitoreo de tablas globales
- Uso de IAM con tablas globales

Tablas globales: cómo funcionan

> **Important**
>
> Esta documentación corresponde a la versión 2017.11.29 (heredada) de las tablas globales, lo que debe evitarse en el caso de las tablas globales nuevas. Los clientes deberían utilizar la versión 2019.11.21 (actual) de las tablas globales siempre que sea posible, ya que proporciona mayor flexibilidad, mayor eficacia y consume menos capacidad de escritura que la 2017.11.29 (heredada).
> Para determinar qué versión utiliza, consulte Determinación de la versión de una tabla global.
> Para actualizar las tablas globales existentes de la versión 2017.11.29 (heredada) a la versión 2019.11.21 (actual), consulte Versiones de tablas globales de DynamoDB.

Las secciones siguientes le ayudarán a comprender los conceptos y el comportamiento de las tablas globales en Amazon DynamoDB.

Conceptos de tablas globales para la versión 2017.11.29 (heredada)

Una tabla global es una colección de una o más réplicas de tabla, propiedad de una única cuenta de AWS.

Una réplica de tabla (o réplica) es una única tabla de DynamoDB que funciona como una parte de una tabla global. Cada réplica almacena el mismo conjunto de elementos de datos. Cualquier tabla global solo puede tener una réplica de tabla por región de AWS.

A continuación se muestra información general de carácter conceptual acerca de cómo se crea una tabla global.

1. Cree una tabla normal de DynamoDB con DynamoDB Streams habilitada en una región AWS.
2. Repita el paso 1 para cada región en la que desea replicar los datos.
3. Defina una tabla global de DynamoDB en función de las tablas que haya creado.

La Consola de administración de AWS automatiza estas tareas para que pueda crear una tabla global de forma más rápida y sencilla. Para obtener más información, consulte Creación de una tabla global.

La tabla global de DynamoDB resultante se compone de varias réplicas de tabla, una por región, que DynamoDB considera como una única unidad. Cada réplica tiene el mismo nombre de tabla y

el mismo esquema de clave primaria. Cuando una aplicación escribe datos en una réplica de tabla de una región, DynamoDB propaga automáticamente la operación de escritura en el resto de las réplicas de tablas de las otras regiones de AWS.

> **Important**
>
> Para mantener los datos de la tabla sincronizados, las tablas globales crean automáticamente los siguientes atributos para cada elemento:
>
> - `aws:rep:deleting`
> - `aws:rep:updatetime`
> - `aws:rep:updateregion`
>
> No modifique estos atributos ni cree atributos con el mismo nombre.

Puede agregar réplicas de tablas a la tabla global para que esté disponible en otras regiones. (Para ello, la tabla global debe estar vacía. En otras palabras, ninguna de las réplicas de tabla puede contener ningún dato).

También puede eliminar una réplica de tabla de una tabla global. En tal caso, la tabla se desasocia completamente de la tabla global. Esta nueva tabla independiente ya no interactúa con la tabla global y los datos ya no se propagarán hacia o desde allí.

> **Warning**
>
> Tenga en cuenta que eliminar una réplica no es un proceso atómico. Para asegurar un comportamiento coherente y un estado conocido, puede considerar la posibilidad de desviar el tráfico de escritura de su aplicación fuera de la réplica que se va a eliminar con antelación. Después de eliminarla, espere hasta que todos los puntos de conexión de la región de réplica muestren la réplica como disociada antes de realizar más escrituras en ella como su propia tabla regional aislada.

Tareas comunes

Las tareas comunes de las tablas globales funcionan de la siguiente manera.

Puede eliminar la tabla de réplica de una tabla global de la misma manera que una tabla normal. Esto detendrá la replicación en esa región y eliminará la copia de la tabla guardada en dicha región. No puede separar la replicación y tener copias de la tabla que existan como entidades independientes.

> **Note**
>
> No podrá eliminar una tabla de origen hasta que transcurran al menos 24 horas desde que se haya utilizado para iniciar una nueva región. Si intenta eliminarla demasiado pronto, se producirá un error.

Pueden surgir conflictos si las aplicaciones actualizan el mismo elemento en diferentes regiones aproximadamente al mismo momento. Para garantizar la coherencia final, las tablas globales de DynamoDB usan el método “el último escritor gana”. Todas las réplicas estarán de acuerdo con la última actualización y convergerán en un estado en el cual todas tienen datos idénticos.

> **Note**
>
> Existen varias maneras de evitar los conflictos, entre las que se incluyen:
>
> - Uso de una política de IAM para permitir solo escrituras en la tabla en una región.
> - Uso de una política de IAM para redirigir a los usuarios a una sola región y mantener la otra en espera inactiva o, alternativamente, redirigir a los usuarios impares a una región y a los usuarios pares a otra.
> - Evitar el uso de actualizaciones no idempotentes, como Bookmark = Bookmark + 1, en favor de actualizaciones estáticas como Bookmark=25.

Monitoreo de tablas globales

Puede utilizar CloudWatch para observar la métrica `ReplicationLatency`. Esta métrica realiza el seguimiento del tiempo transcurrido entre el momento en que un elemento aparece en la secuencia de DynamoDB para una tabla de réplica y el momento en que dicho elemento aparece en otra réplica de la tabla global. `ReplicationLatency` se expresa en milisegundos y se emite para cada pareja de región de origen y región de destino. Esta es la única métrica de CloudWatch que proporciona la versión 2 de las tablas globales.

Las latencias que observará dependerán de la distancia entre las regiones elegidas, así como de otras variables. Las latencias en el intervalo de 0,5 a 2,5 segundos para las regiones pueden ser comunes en la misma área geográfica.

Tiempo de vida (TTL)

Puede utilizar el tiempo de vida (TTL) para especificar un nombre de atributo cuyo valor indique el tiempo de caducidad del elemento. Este valor se especifica como un número en segundos desde el inicio del tiempo Unix.

Con la versión heredada de las tablas globales, las eliminaciones de TTL no se replican automáticamente en otras réplicas. Cuando se elimina un elemento mediante una regla de TTL, ese trabajo se realiza sin consumir unidades de escritura.

Tenga en cuenta que si las tablas de origen y destino tienen una capacidad de escritura aprovisionada muy baja, se puede producir una limitación, ya que las eliminaciones de TTL requieren capacidad de escritura.

Flujos y transacciones con tablas globales

Cada tabla global produce un flujo independiente basado en todas sus escrituras, independientemente del punto de origen de dichas escrituras. Puede optar por consumir este flujo de DynamoDB en una región o en todas las regiones de forma independiente.

Si desea procesar escrituras locales pero no escrituras replicadas, puede agregar su propio atributo de región a cada elemento. A continuación, puede utilizar un filtro de eventos de Lambda para invocar únicamente a Lambda para las escrituras en la región local.

Las operaciones transaccionales proporcionan garantías ACID (atomicidad, coherencia, aislamiento y durabilidad) SOLO en la región en la que se crea la escritura originalmente. No se admiten las transacciones entre regiones en las tablas globales.

Por ejemplo, si tiene una tabla global con réplicas en las regiones Este de EE. UU. (Ohio) y Oeste de EE. UU. (Oregón) y realiza una operación TransactWriteItems en la región Este de EE. UU. (Ohio), puede observar transacciones completadas parcialmente en la región Oeste de EE. UU. (Oregón) a medida que los cambios se replican. Solo se replicarán los cambios en otras regiones cuando se hayan confirmado en la región de origen.

> **Note**
>
> - Las tablas globales “sustituyen” a Acelerador de DynamoDB al actualizar DynamoDB directamente. Como resultado, DAX no sabrá que contiene datos obsoletos. La memoria caché de DAX solo se actualizará cuando caduque el TTL de la memoria caché.
> - Las etiquetas de las tablas globales no se propagan automáticamente.

Rendimiento de lectura y escritura

Las tablas globales administran el rendimiento de lectura y escritura de las siguientes maneras.

- La capacidad de escritura debe ser la misma en todas las instancias de tabla en todas las regiones.
- Con la versión 2019.11.21 (actual), si la tabla está configurada para admitir el escalado automático o está en el modo bajo demanda, la capacidad de escritura se mantiene sincronizada automáticamente. La cantidad actual de capacidad de escritura aprovisionada en cada región aumentará y disminuirá de forma independiente según la configuración de escalado automático sincronizado. Si la tabla se coloca en modo bajo demanda, ese modo se sincronizará con las demás réplicas.
- La capacidad de lectura puede diferir de una región a otra porque las lecturas pueden no ser iguales. Al agregar una réplica global a una tabla, se propaga la capacidad de la región de origen. Tras la creación, puede ajustar la capacidad de lectura de una réplica y esta nueva configuración no se transferirá al otro lado.

Coherencia y resolución de conflictos

Cualquier cambio en cualquier elemento de cualquier réplica de tabla se replicará en el resto de réplicas en la misma tabla global. En una tabla global, un elemento que se acaba de escribir se propagará normalmente a todas las réplicas de tabla en cuestión de segundos.

Con una tabla global, cada réplica de tabla almacena el mismo conjunto de elementos de datos. DynamoDB no admite la replicación parcial de solo algunos de los elementos.

Una aplicación puede leer y escribir datos en cualquier réplica de tabla. DynamoDB admite lecturas coherentes posteriores entre regiones, pero no admite lecturas altamente coherentes entre regiones. Si la aplicación solo utiliza operaciones de lectura eventualmente consistentes y solo realiza

operaciones de lectura en una región de AWS, funcionará sin ninguna modificación. No obstante, si su aplicación requiere lecturas altamente coherentes, debe realizar todas las lecturas y escrituras altamente coherentes en la misma región. De no ser así, si realiza una escritura en una región y una lectura en otra, es posible que la respuesta de la lectura incluya datos anticuados que no reflejen los resultados de las operaciones de escritura completadas recientemente en la otra región.

Pueden surgir conflictos si las aplicaciones actualizan el mismo elemento en diferentes regiones aproximadamente al mismo momento. Para garantizar la consistencia final, las tablas globales de DynamoDB usan la reconciliación gana quien escribe último entre las actualizaciones simultáneas. DynamoDB hace todo lo posible para determinar quién realizó la última escritura. Con este mecanismo de resolución de conflictos, todas las réplicas estarán de acuerdo con la última actualización y convergerán en un estado en el cual todas tienen datos idénticos.

Disponibilidad y durabilidad

Si una única región de AWS se encuentra aislada o degradada, la aplicación puede redirigir a otra región y realizar las operaciones de lectura y escritura en una réplica de tabla diferente. Puede aplicar una lógica empresarial personalizada para determinar cuándo deben redirigirse solicitudes a otras regiones.

Si una región se encuentra aislada o degradada, DynamoDB lleva un seguimiento de las operaciones de escritura que se han realizado pero que todavía no se han propagado a todas las réplicas de tabla. Cuando la región vuelva a estar en línea, DynamoDB reanudará la propagación de cualquier operación de escritura pendiente desde esa región a las réplicas de tabla en otras regiones. Asimismo, reanudará la propagación de las operaciones de escritura de otras réplicas de tabla a la región que ahora está en línea. Todas las escrituras realizadas correctamente con anterioridad se propagarán finalmente sin importar el tiempo que la región permanezca aislada.

Prácticas recomendadas y requisitos para la administración de tablas globales

> **Important**
>
> Esta documentación corresponde a la versión 2017.11.29 (heredada) de las tablas globales, lo que debe evitarse en el caso de las tablas globales nuevas. Los clientes deberían utilizar la versión 2019.11.21 (actual) de las tablas globales siempre que sea posible, ya que proporciona mayor flexibilidad, mayor eficacia y consume menos capacidad de escritura que la 2017.11.29 (heredada).

Para determinar qué versión utiliza, consulte Determinación de la versión de una tabla global. Para actualizar las tablas globales existentes de la versión 2017.11.29 (heredada) a la versión 2019.11.21 (actual), consulte Versiones de tablas globales de DynamoDB.

Mediante las tablas globales de Amazon DynamoDB, puede replicar los datos de la tabla en las regiones de AWS. Es importante que las réplicas de tabla y los índices secundarios de la tabla global tengan una configuración de capacidad de escritura idéntica para garantizar la replicación adecuada de los datos.

Temas

- Versión de tablas globales
- Requisitos para agregar una tabla de réplica nueva
- Prácticas recomendadas y requisitos para administrar la capacidad

Versión de tablas globales

Hay disponibles dos versiones de las tablas globales de DynamoDB: versión 2019.11.21 (actual) de las tablas globales y Versión 2017.11.29 (heredada) de las tablas globales. Los clientes deberían utilizar la versión 2019.11.21 (actual) de las tablas globales siempre que sea posible, ya que proporciona mayor flexibilidad, mayor eficacia y consume menos capacidad de escritura que la 2017.11.29 (heredada).

Para determinar qué versión utiliza, consulte Determinación de la versión de una tabla global. Para actualizar las tablas globales existentes de la versión 2017.11.29 (heredada) a la versión 2019.11.21 (actual), consulte Actualización de tablas globales.

Requisitos para agregar una tabla de réplica nueva

Si quiere agregar un tabla de réplica nueva a una tabla global, debe cumplirse cada una de las condiciones siguientes:

- La tabla debe tener la misma clave de partición que todas las demás réplicas.
- La tabla debe tener especificada la misma configuración de administración de capacidad de escritura.
- La tabla debe tener el mismo nombre que todas las demás réplicas.

- La tabla debe tener DynamoDB Streams habilitado y la secuencia debe contener las imágenes viejas y las nuevas del elemento.
- Ninguna de las réplicas de tabla nueva o que ya existe en la tabla global puede contener ningún dato.

Si se especifican índices secundarios globales, también se deben cumplir las siguientes condiciones:

- Los índices secundarios globales deben tener el mismo nombre.
- Los índices secundarios globales deben tener la misma clave de partición y clave de clasificación (si procede).

> **Important**
>
> La configuración de capacidad de escritura debe establecerse de forma coherente en todas las réplicas de tabla globales y en los índices secundarios coincidentes. Para actualizar la configuración de capacidad de escritura de la tabla global, recomendamos encarecidamente utilizar la consola de DynamoDB o la operación de la API `UpdateGlobalTableSettings`. `UpdateGlobalTableSettings` aplica cambios a la configuración de capacidad de escritura a todas las réplicas de tabla y a los índices secundarios coincidentes de una tabla global automáticamente. Si utiliza las operaciones `UpdateTable`, `RegisterScalableTarget` o `PutScalingPolicy`, debe aplicar el cambio a cada réplica de tabla y al índice secundario coincidente individualmente. Para obtener más información, consulte UpdateGlobalTableSettings en la Referencia de la API de Amazon DynamoDB. Recomendamos encarecidamente que habilite la función Auto Scaling para administrar la configuración de capacidad de escritura aprovisionada. Si prefiere administrar manualmente la configuración de capacidad de escritura, debe aprovisionar unidades de capacidad de escritura replicada iguales a todas las tablas de réplica. Aprovisione también unidades de capacidad de escritura replicada iguales para coincidencias de índices secundarios en toda la tabla global.
>
> A su vez, también debe contar con los permisos de AWS Identity and Access Management(IAM) adecuados. Para obtener más información, consulte Uso de IAM con tablas globales.

Prácticas recomendadas y requisitos para administrar la capacidad

Tenga en cuenta lo siguiente al administrar la configuración de capacidad para réplicas de tablas en DynamoDB.

Uso de Auto Scaling de DynamoDB

El uso de Auto Scaling de DynamoDB es la manera recomendada de administrar la configuración de capacidad de rendimiento para las réplicas de tabla que utilizan el modo aprovisionado. Auto Scaling de DynamoDB ajusta automáticamente las unidades de capacidad de lectura (RCU) y las unidades de capacidad de escritura (WCU) en cada réplica de tabla, en función de la carga de trabajo de la aplicación real. Para obtener más información, consulte Administración automática de la capacidad de rendimiento con la función Auto Scaling de DynamoDB.

Si crea sus tablas de réplica mediante la Consola de administración de AWS, se habilitará Auto Scaling de forma predeterminada para cada réplica de tabla con la configuración de Auto Scaling predeterminada para la administración de unidades de capacidad de lectura y escritura.

Los cambios en la configuración de Auto Scaling para una réplica de tabla o índice secundario realizados a través de la consola de DynamoDB o mediante la llamada `UpdateGlobalTableSettings` se aplican automáticamente a todas las réplicas de tabla y a los índices secundarios coincidentes de la tabla global. Estos cambios sobrescriben cualquier configuración de Auto Scaling existente. Esto garantiza que la configuración de capacidad de escritura aprovisionada sea coherente entre las réplicas de tabla y los índices secundarios de la tabla global. Si utiliza las llamadas `UpdateTable`, `RegisterScalableTarget` o `PutScalingPolicy`, debe aplicar el cambio a cada réplica de tabla y al índice secundario coincidente individualmente.

> **Note**
>
> Si el Auto Scaling no satisface los cambios de capacidad de la aplicación (carga de trabajo impredecible) o si no desea configurar sus opciones (configuración de destino para el umbral mínimo, máximo o de utilización), puede usar el modo bajo demanda para administrar la capacidad de las tablas globales. Para obtener más información, consulte Modo bajo demanda.
>
> Si habilita el modo bajo demanda en una tabla global, el consumo de unidades de solicitud de escritura replicada (RWCU) será coherente con la forma en que se aprovisionan las RCU. Por ejemplo, si realiza 10 escrituras en una tabla local que se replica en dos regiones adicionales, consumirá 60 unidades de solicitud de escritura (10 + 10 + 10 = 30; 30 x 2 = 60). Las 60 unidades de solicitud de escritura consumidas incluyen la escritura

adicional que consumen las tablas globales versión 2017.11.29 para actualizar los atributos `aws:rep:deleting`, `aws:rep:updatetime` y `aws:rep:updateregion`.

Administración manual de la capacidad

Si decide no utilizar Auto Scaling de DynamoDB, tendrá que ajustar manualmente la configuración de la capacidad de lectura y escritura en cada réplica de tabla e índice secundario.

Las RWCU aprovisionadas en cada réplica de tabla deben establecerse en el número total de RWCU necesarias para las escrituras de aplicaciones en todas las regiones multiplicadas por dos. Esto adapta las escrituras de aplicaciones que se producen en la región local y las escrituras de aplicaciones replicadas procedentes de otras regiones. Por ejemplo, supongamos que espera 5 escrituras por segundo en la réplica de tabla en Ohio y 5 escrituras por segundo en la réplica de tabla en el Norte de Virginia. En este caso, debe aprovisionar 20 rWCU para cada réplica de tabla (5 + 5 = 10; 10 x 2 = 20).

Para actualizar la configuración de capacidad de escritura de la tabla global, recomendamos encarecidamente utilizar la consola de DynamoDB o la operación de la API `UpdateGlobalTableSettings`. `UpdateGlobalTableSettings` aplica cambios a la configuración de capacidad de escritura a todas las réplicas de tabla y a los índices secundarios coincidentes de una tabla global automáticamente. Si utiliza las operaciones `UpdateTable`, `RegisterScalableTarget` o `PutScalingPolicy`, debe aplicar el cambio a cada réplica de tabla y al índice secundario coincidente individualmente. Para obtener más información, consulte la Referencia de las API de Amazon DynamoDB.

> **Note**
>
> Para actualizar la configuración (`UpdateGlobalTableSettings`) para una tabla global en DynamoDB, debe tener los permisos `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy` y `application-autoscaling:DeregisterScalableTarget`. Para obtener más información, consulte Uso de IAM con tablas globales.

Creación de una tabla global

> **Important**
>
> Esta documentación corresponde a la versión 2017.11.29 (heredada) de las tablas globales, lo que debe evitarse en el caso de las tablas globales nuevas. Los clientes deberían utilizar la versión 2019.11.21 (actual) de las tablas globales siempre que sea posible, ya que proporciona mayor flexibilidad, mayor eficacia y consume menos capacidad de escritura que la 2017.11.29 (heredada).
> Para determinar qué versión utiliza, consulte Determinación de la versión de una tabla global.
> Para actualizar las tablas globales existentes de la versión 2017.11.29 (heredada) a la versión 2019.11.21 (actual), consulte Versiones de tablas globales de DynamoDB.

En esta sección se indica cómo crear una tabla global mediante la consola de Amazon DynamoDB o la AWS Command Line Interface (AWS CLI).

Temas

- Creación de una tabla global (consola)
- Creación de una tabla global (AWS CLI)

Creación de una tabla global (consola)

Siga estos pasos para crear una tabla global mediante la consola. En el siguiente ejemplo se crea una tabla global con tablas de réplica en los Estados Unidos y Europa.

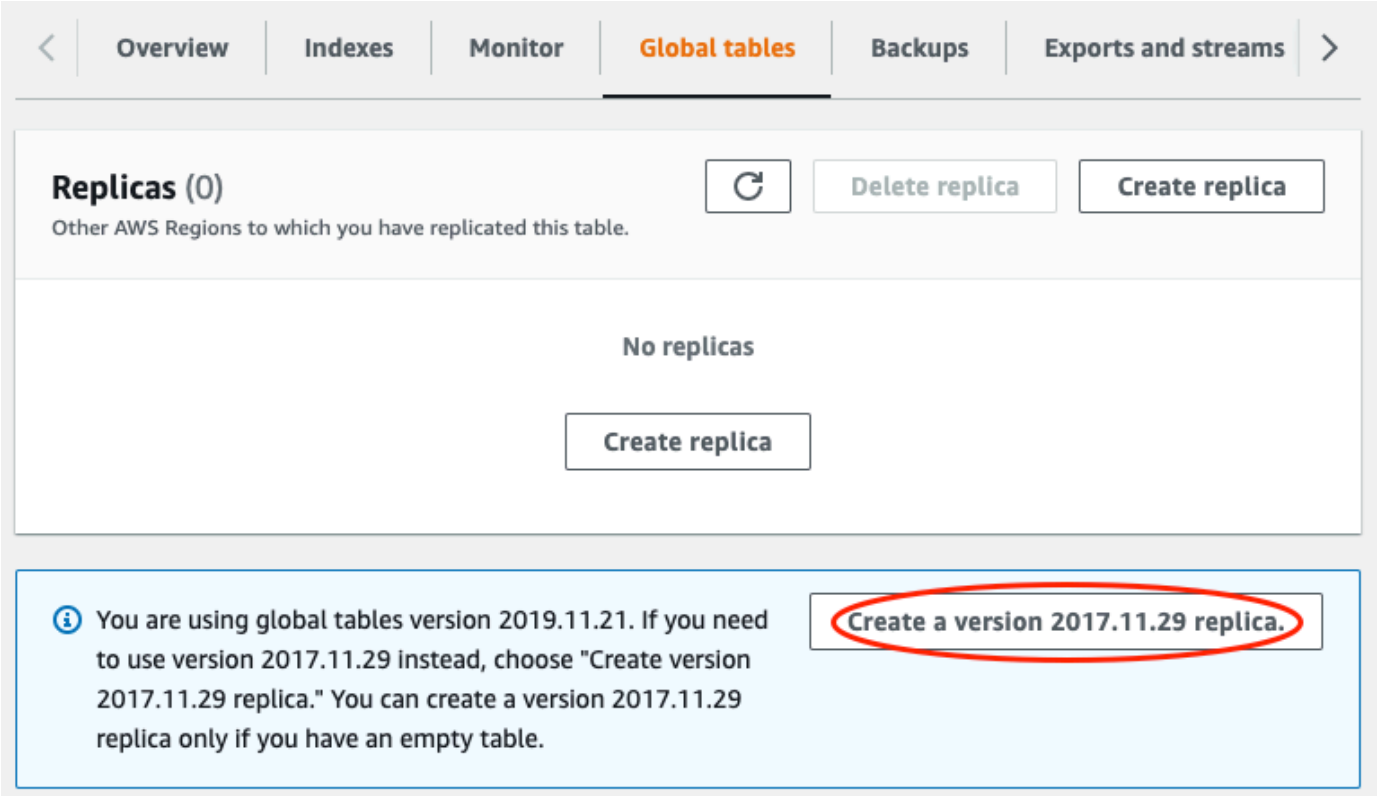
1. Abra la consola de DynamoDB en https://console.aws.amazon.com/dynamodb/home. Para realizar este ejemplo, elija la región us-east-2 (EE. UU. Este [Ohio]).
2. En el panel de navegación del lado izquierdo de la consola, elija Tablas.
3. Seleccione Create Table (Crear tabla).

   En Nombre de la tabla, introduzca **`Music`**.

   En Primary key (Clave principal) introduzca **`Artist`**. Elija Add sort key (Añadir clave de ordenación) e introduzca **`SongTitle`**. (Tanto **`Artist`** como **`SongTitle`** deben ser cadenas).

Para crear la tabla, seleccione Create (Crear). Esta tabla le servirá a modo de primera tabla de réplica en una nueva tabla global. Será el prototipo para crear otras tablas de réplica que quiera añadir más tarde.

4. Elija la pestaña Tablas globales y, a continuación, elija Crear una réplica de la versión 2017.11.29 (heredada).



5. En el menú desplegable Available replication Regions (Regiones de replicación disponibles), elija US West (Oregon) (Oeste de EE. UU. (Oregón).

   La consola comprueba el proceso para asegurarse de que no haya ninguna tabla con el mismo nombre en la región seleccionada. Si ya existe una tabla con el mismo nombre, debe eliminar la tabla existente antes de crear una nueva tabla de réplica en esa región.

6. Elija Create replica (Crear réplica). Esto comienza el proceso de creación de la tabla en Oeste de EE. UU (Oregón).

   La pestaña Global Table (Tabla global) de la tabla seleccionada (y de cualquier otra tabla de réplica) indica que la tabla se ha replicado en varias regiones.

7. Ahora puede añadir otra región para que la tabla global se replique y sincronice en los Estados Unidos y Europa. Para ello, repita el paso 5, pero esta vez especifique Europe (Fráncfort) (Europa (Fráncfort) en lugar de US West (Oregon) (Oeste de EE. UU. (Oregón)).

8. Debe seguir usando la Consola de administración de AWS en la región (Este de EE. UU. (Ohio)). Seleccione Items (Elementos) en el menú de navegación izquierdo, seleccione la tabla Music (Música) y, a continuación, elija Create Item (Crear elemento).

   a. En Artist (Artista), escriba **`item_1`**.

   b. En SongTitle, escriba **`Song Value 1`**.

   c. Para escribir el elemento, elija Create item (Crear elemento).

9. Después de un breve periodo de tiempo, el elemento se replica en las tres regiones de la tabla global. Para comprobar esto, en la consola, vaya al selector de regiones que se encuentra en la esquina superior derecha y elija Europa (Frankfurt). La tabla `Music` de Europa (Frankfurt) debe contener el nuevo elemento.

10. Repita el paso 9 y elija US West (Oregon) (Oeste de EE. UU. [Oregón]) para verificar la replicación en esa región.

Creación de una tabla global (AWS CLI)

Siga estos pasos para crear una tabla global `Music` mediante la AWS CLI. En el siguiente ejemplo se crea una tabla global con tablas de réplica en los Estados Unidos y en Europa.

1. Cree una nueva tabla (`Music`) en EE. UU. Este (Ohio) con DynamoDB Streams habilitada (`NEW_AND_OLD_IMAGES`).

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-2
```

2. Cree un tabla `Music` idéntica en EE. UU. Este (Norte de Virginia).

```
aws dynamodb create-table \
    --table-name Music \
```

```
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region us-east-1
```

3. Cree una tabla global (`Music`) que se componga de réplicas de tablas en las regiones `us-east-2` y `us-east-1`.

```
aws dynamodb create-global-table \
    --global-table-name Music \
    --replication-group RegionName=us-east-2 RegionName=us-east-1 \
    --region us-east-2
```

> **Note**
>
> El nombre de la tabla global (`Music`) debe coincidir con el nombre de cada una de las réplicas de tabla (`Music`). Para obtener más información, consulte Prácticas recomendadas para tablas globales.

4. Cree otra tabla en Europa (Irlanda), con la misma configuración que usó en los pasos 1 y 2:

```
aws dynamodb create-table \
    --table-name Music \
    --attribute-definitions \
        AttributeName=Artist,AttributeType=S \
        AttributeName=SongTitle,AttributeType=S \
    --key-schema \
        AttributeName=Artist,KeyType=HASH \
        AttributeName=SongTitle,KeyType=RANGE \
    --provisioned-throughput \
        ReadCapacityUnits=10,WriteCapacityUnits=5 \
    --stream-specification StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES \
    --region eu-west-1
```

Después de realizar este paso, agregue la nueva tabla a la tabla global `Music`.

```
aws dynamodb update-global-table \
    --global-table-name Music \
    --replica-updates 'Create={RegionName=eu-west-1}' \
    --region us-east-2
```

5. Para verificar que la replicación funciona, agregue un nuevo elemento a la tabla `Music` en EE. UU. Este (Ohio).

```
aws dynamodb put-item \
    --table-name Music \
    --item '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-2
```

6. Espere unos segundos y, a continuación, verifique si el elemento se replicó correctamente en las regiones EE. UU. Este (Norte de Virginia) y Europa (Irlanda).

```
aws dynamodb get-item \
    --table-name Music \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region us-east-1
```

```
aws dynamodb get-item \
    --table-name Music \
    --key '{"Artist": {"S":"item_1"},"SongTitle": {"S":"Song Value 1"}}' \
    --region eu-west-1
```

Monitoreo de tablas globales

> **Important**
>
> Esta documentación corresponde a la versión 2017.11.29 (heredada) de las tablas globales, lo que debe evitarse en el caso de las tablas globales nuevas. Los clientes deberían utilizar la versión 2019.11.21 (actual) de las tablas globales siempre que sea posible, ya que proporciona mayor flexibilidad, mayor eficacia y consume menos capacidad de escritura que la 2017.11.29 (heredada).

Para determinar qué versión utiliza, consulte Determinación de la versión de una tabla global. Para actualizar las tablas globales existentes de la versión 2017.11.29 (heredada) a la versión 2019.11.21 (actual), consulte Versiones de tablas globales de DynamoDB.

Puede utilizar Amazon CloudWatch para monitorear el comportamiento y el rendimiento de una tabla global. Amazon DynamoDB publica las métricas `ReplicationLatency` y `PendingReplicationCount` para cada réplica en la tabla global.

- **`ReplicationLatency`** El tiempo transcurrido entre el momento en que un elemento se escribe en la transmisión de la réplica de tabla de DynamoDB y el momento en que dicho elemento aparece en otra réplica de la tabla global. `ReplicationLatency` se expresa en milisegundos y se emite para cada pareja de región de origen y destino.

  Durante el uso normal, el valor de `ReplicationLatency` debería ser bastante constante. Un valor alto de `ReplicationLatency` podría indicar que las actualizaciones de una réplica no se están propagando hacia otras tablas de réplica de manera puntual. Con el tiempo, esto podría dar lugar a que otras tablas de réplica se quedaran rezagadas, ya que dejarían de recibir actualizaciones de forma consistente. En este caso, debería verificar que las unidades de capacidad de lectura (RCU) y de escritura (WCU) son idénticas para cada una de las tablas de réplica. Además, al elegir la configuración de WCU, debe seguir las recomendaciones de Versión de tablas globales.

  El valor de `ReplicationLatency` puede aumentar si una región de AWS se encuentra degradada y tiene una réplica de tabla en esa región. En este caso, puede redirigir temporalmente la actividad de lectura y escritura de la aplicación a otra región de AWS.

- **`PendingReplicationCount`**: el número de actualizaciones de elementos que se escriben en una réplica de tabla pero que aún no se han escrito en otra réplica de la tabla global. `PendingReplicationCount` se expresa en el número de elementos y se emite para cada uno de los pares de región de origen y destino.

  Durante el uso normal, el valor de `PendingReplicationCount` debería ser muy bajo. Si el valor de `PendingReplicationCount` aumenta durante periodos prolongados de tiempo, debería investigar si la configuración de la capacidad de escritura que aprovisionó su réplica de tabla es suficiente para la carga de trabajo actual.

El valor de `PendingReplicationCount` puede aumentar si una región de AWS se encuentra degradada y tiene una réplica de tabla en esa región. En este caso, puede redirigir temporalmente la actividad de lectura y escritura de la aplicación a otra región de AWS.

Para obtener más información, consulte Dimensiones y métricas de DynamoDB.

Uso de IAM con tablas globales

> **Important**
>
> Esta documentación corresponde a la versión 2017.11.29 (heredada) de las tablas globales, lo que debe evitarse en el caso de las tablas globales nuevas. Los clientes deberían utilizar la versión 2019.11.21 (actual) de las tablas globales siempre que sea posible, ya que proporciona mayor flexibilidad, mayor eficacia y consume menos capacidad de escritura que la 2017.11.29 (heredada).
> Para determinar qué versión utiliza, consulte Determinación de la versión de una tabla global.
> Para actualizar las tablas globales existentes de la versión 2017.11.29 (heredada) a la versión 2019.11.21 (actual), consulte Versiones de tablas globales de DynamoDB.

Cuando se crea una tabla global por primera vez, Amazon DynamoDB crea automáticamente un rol vinculado al servicio AWS Identity and Access Management (IAM). Este rol se denomina y permite que DynamoDB administre la replicación entre regiones de las tablas globales en su nombre. No elimine este rol vinculado a un servicio. Si lo hace, las tablas globales no funcionarán.

Para obtener más información acerca los roles vinculados a servicios, consulte Uso de roles vinculados a servicios en la Guía del usuario de IAM.

Para crear y mantener las tablas globales en DynamoDB, debe contar con el permiso de `dynamodb:CreateGlobalTable` para obtener acceso a cada uno de los siguientes elementos:

- La réplica de tabla que quiera agregar.
- Cada réplica existente que ya forme parte de la tabla global.
- La misma tabla global.

Para actualizar la configuración (`UpdateGlobalTableSettings`) para una tabla global en DynamoDB, debe tener los permisos `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy` y `application-autoscaling:DeregisterScalableTarget`.

Se requieren los permisos `application-autoscaling:DeleteScalingPolicy` y `application-autoscaling:DeregisterScalableTarget` cuando se actualiza una política de escalado existente. Esto es para que el servicio de tablas globales pueda eliminar la política de escalado anterior antes de adjuntar la nueva política a la tabla o al índice secundario.

Si usa una política de IAM para administrar el acceso a una réplica de tabla, deberá aplicar una política idéntica al resto de réplicas de dicha tabla global. De este modo, podrá mantener un modelo de permisos coherente en todas las réplicas de tablas.

Al usar políticas de IAM idénticas en todas las réplicas de una tabla global, puede evitar que se conceda inintencionadamente acceso de lectura y escritura a la tabla global de datos. Pongamos como ejemplo a un usuario que obtenga acceso a una única réplica de una tabla global. Si tal usuario puede escribir en esa réplica, DynamoDB propagará lo que haya escrito al resto de las réplicas de tabla. En efecto, el usuario puede escribir (indirectamente) en el resto de réplicas de la tabla global. Esta situación se puede evitar si usa políticas de IAM coherentes en todas las tablas de réplica.

Ejemplo: permitir la acción CreateGlobalTable

Antes de agregar una réplica a una tabla global, debe contar con el permiso de `dynamodb:CreateGlobalTable` de la tabla global y de cada una de sus tablas de réplica.

La siguiente política de IAM concede permiso para llevar a cabo la acción `CreateGlobalTable` en todas las tablas.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["dynamodb:CreateGlobalTable"],
            "Resource": "*"
```

```
        }
    ]
}
```

Ejemplo: permitir las acciones UpdateGlobalTable, DescribeLimits, application-autoscaling:DeleteScalingPolicy y application-autoscaling:DeregisterScalableTarget

Para actualizar la configuración (`UpdateGlobalTableSettings`) para una tabla global en DynamoDB, debe tener los permisos `dynamodb:UpdateGlobalTable`, `dynamodb:DescribeLimits`, `application-autoscaling:DeleteScalingPolicy` y `application-autoscaling:DeregisterScalableTarget`.

La siguiente política de IAM concede permiso para llevar a cabo la acción `UpdateGlobalTableSettings` en todas las tablas.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:UpdateGlobalTable",
                "dynamodb:DescribeLimits",
                "application-autoscaling:DeleteScalingPolicy",
                "application-autoscaling:DeregisterScalableTarget"
            ],
            "Resource": "*"
        }
    ]
}
```

Ejemplo: permitir la acción CreateGlobalTable para un nombre de tabla global específico con réplicas permitidas solo en ciertas regiones

La siguiente política de IAM concede permisos para permitir la acción `CreateGlobalTable` para crear una tabla global denominada `Customers` con réplicas en dos regiones.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:CreateGlobalTable",
            "Resource": [
                "arn:aws:dynamodb::123456789012:global-table/Customers",
                "arn:aws:dynamodb:us-east-1:123456789012:table/Customers",
                "arn:aws:dynamodb:us-west-1:123456789012:table/Customers"
            ]
        }
    ]
}
```

Versión anterior de la API de DynamoDB de bajo nivel (2011-12-05)

En esta sección se documentan las operaciones disponibles en la versión anterior de la API de bajo nivel de DynamoDB (2011-12-05). Esta versión del API de bajo nivel se mantiene para ofrecer compatibilidad retroactiva con las aplicaciones existentes.

Las nuevas aplicaciones deben utilizar la versión actual de la API (2012-08-10). Para obtener más información, consulte Referencia de la API de DynamoDB.

> **Note**
>
> Recomendamos migrar las aplicaciones existentes a la versión más reciente de la API (2012-08-10), porque las nuevas características de DynamoDB no se podrán aplicar retroactivamente en la versión anterior de la API.

Temas

- BatchGetItem
- BatchWriteItem

- CreateTable
- DeleteItem
- DeleteTable
- DescribeTables
- GetItem
- ListTables
- PutItem
- Consultar
- Examen
- UpdateItem
- UpdateTable

BatchGetItem

> **Important**
>
> ***Esta sección se refiere a la versión 2011-12-05 del API, que está obsoleta y no debe utilizarse para nuevas aplicaciones.***
>
> Para consultar la documentación sobre la API de bajo nivel actual, consulte la Referencia de la API de Amazon DynamoDB.

Descripción

La operación `BatchGetItem` devuelve los atributos de varios elementos de varias tablas mediante sus claves primarias. El número máximo de elementos que pueden recuperarse en una única operación es de 100. Además, el número de elementos recuperados está restringido por un límite de tamaño de 1 MB. Si se supera el límite de tamaño de respuesta o se devuelve un resultado parcial por haberse excedido el rendimiento aprovisionado de la tabla o debido a un error de procesamiento interno, DynamoDB devuelve un valor `UnprocessedKeys` para que pueda reintentar la operación comenzando por el siguiente elemento que desea obtener. DynamoDB ajusta automáticamente el número de elementos que se devuelven por página para aplicar este límite. Por ejemplo, aunque solicite que se recuperen 100 elementos con un tamaño individual de 50 KB cada uno, el sistema devolverá 20 elementos y un valor de `UnprocessedKeys` correspondiente para que pueda obtener

la siguiente página de resultados. Si lo desea, la aplicación puede incluir su propia lógica para ensamblar las páginas de resultados en un único conjunto.

Si no se puede procesar ningún elemento porque no hay rendimiento aprovisionado suficiente en cada una de las tablas objeto de la solicitud, DynamoDB devuelve un error `ProvisionedThroughputExceededException`.

> **Note**
>
> De forma predeterminada, `BatchGetItem` realiza lecturas consistentes finales en cada tabla incluida en la solicitud. Si desea realizar en su lugar lecturas consistentes, puede establecer el parámetro `ConsistentRead` en `true` para cada tabla.
> `BatchGetItem` recupera los elementos en paralelo para minimizar las latencias de respuesta.
> Al diseñar la aplicación, tenga en cuenta que DynamoDB no garantiza cómo se ordenarán los atributos en la respuesta devuelta. Incluya en `AttributesToGet` los valores de clave principal de los elementos de la solicitud para ayudar a analizar la respuesta por elementos.
> Si los elementos solicitados no existen, no se devuelve nada en la respuesta para esos elementos. Las solicitudes de elementos inexistentes consumen las unidades de capacidad de lectura mínimas, según el tipo de lectura. Para obtener más información, consulte Tamaños y formatos de elementos de DynamoDB.

Solicitudes

Sintaxis

```
// This header is abbreviated. For a sample of a complete header, see API de bajo nivel
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{"RequestItems":
    {"Table1":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue1"}, "RangeKeyElement":
{"N":"KeyValue2"}},
            {"HashKeyElement": {"S":"KeyValue3"}, "RangeKeyElement":{"N":"KeyValue4"}},
```

```
            {"HashKeyElement": {"S":"KeyValue5"}, "RangeKeyElement":
{"N":"KeyValue6"}}],
        "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
    "Table2":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}}],
        "AttributesToGet": ["AttributeName4", "AttributeName5", "AttributeName6"]
        }
    }
}
```

Nombre	Descripción	Obligatorio
`RequestItems`	Un contenedor del nombre de la tabla y los elementos correspondientes que se deben obtener según su clave principal. Mientras se solicitan elementos, cada nombre de tabla puede invocarse solo una vez por operación. Tipo: cadena Valor predeterminado: None	Sí
`Table`	El nombre de la tabla que contiene los elementos que hay que obtener. La entrada es simplemente una cadena que especifica una tabla existente sin ninguna etiqueta. Tipo: cadena Valor predeterminado: None	Sí
`Table:Keys`	Los valores de clave principal que definen los elementos	Sí

Nombre	Descripción	Obligatorio
	de la tabla especificada. Para obtener más informaci ón sobre claves principales, consulte Clave principal. Tipo: Keys	
`Table:AttributesToGet`	Matriz de nombres de atributos contenidos en la tabla especificada. Si no se especifican sus nombres, se devuelven todos los atributos . Si algún atributo no se encuentra, no aparecerá en los resultados. Tipo: matriz	No
`Table:ConsistentRead`	Si se establece en `true`, se emite una lectura consisten te; en caso contrario, se utiliza una lectura consistente final. Tipo: Booleano	No

Respuestas

Sintaxis

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 855

{"Responses":
    {"Table1":
        {"Items":
        [{"AttributeName1": {"S":"AttributeValue"},
```

```
        "AttributeName2": {"N":"AttributeValue"},
        "AttributeName3": {"SS":["AttributeValue", "AttributeValue", "AttributeValue"]}
        },
        {"AttributeName1": {"S": "AttributeValue"},
        "AttributeName2": {"S": "AttributeValue"},
        "AttributeName3": {"NS": ["AttributeValue", "AttributeValue",
 "AttributeValue"]}
        }],
    "ConsumedCapacityUnits":1},
    "Table2":
        {"Items":
        [{"AttributeName1": {"S":"AttributeValue"},
        "AttributeName2": {"N":"AttributeValue"},
        "AttributeName3": {"SS":["AttributeValue", "AttributeValue", "AttributeValue"]}
        },
        {"AttributeName1": {"S": "AttributeValue"},
        "AttributeName2": {"S": "AttributeValue"},
        "AttributeName3": {"NS": ["AttributeValue", "AttributeValue","AttributeValue"]}
        }],
    "ConsumedCapacityUnits":1}
    },
    "UnprocessedKeys":
        {"Table3":
        {"Keys":
            [{"HashKeyElement": {"S":"KeyValue1"}, "RangeKeyElement":
{"N":"KeyValue2"}},
            {"HashKeyElement": {"S":"KeyValue3"}, "RangeKeyElement":{"N":"KeyValue4"}},
            {"HashKeyElement": {"S":"KeyValue5"}, "RangeKeyElement":
{"N":"KeyValue6"}}],
        "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]}
        }
}
```

Nombre	Descripción
`Responses`	Nombres de tablas y los atributos de los elementos respectivos de las tablas. Tipo: mapa
`Table`	El nombre de la tabla que contiene los elementos. La entrada es simplemente una

Nombre	Descripción
	cadena que especifica la tabla existente sin ninguna etiqueta. Tipo: cadena
`Items`	Contenedor de los nombres y valores de los atributos que coinciden con los parámetros de la operación. Tipo: Map, mapa de los nombres de atributos y sus tipos de datos y valores.
`ConsumedCapacityUnits`	La cantidad de unidades de capacidad de lectura consumidas para cada tabla. Este valor muestra el número aplicado al rendimiento aprovisionado. Las solicitudes de elementos inexistentes consumen las unidades de capacidad de lectura mínimas, según el tipo de lectura. Para obtener más información, consulte Modo de capacidad aprovisionada de DynamoDB. Tipo: Number
`UnprocessedKeys`	Contiene una matriz de tablas y sus respectivas claves que no se han procesado en la respuesta actual, posiblemente debido a que se ha alcanzado un límite de tamaño de respuesta. El valor de `UnprocessedKeys` tiene el mismo formato que el parámetro `RequestItems` (por lo que se puede proporcionar directamente a una operación `BatchGetItem` posterior). Para obtener más información, consulte el parámetro `RequestItems` anterior. Tipo: matriz

Nombre	Descripción
`UnprocessedKeys` : `Table`: `Keys`	Los valores de los atributos de clave principal que definen los elementos y los atributos asociados con los elementos. Para obtener más información sobre claves principales, consulte Clave principal. Tipo: Array; matriz de pares de nombre-valor del atributo.
`UnprocessedKeys` : `Table`: `AttributesToGet`	Nombres de los atributos contenidos en la tabla especificada. Si no se especifican sus nombres, se devuelven todos los atributos. Si algún atributo no se encuentra, no aparecerá en los resultados. Tipo: Array, matriz de nombres de atributos.
`UnprocessedKeys` : `Table`: `ConsistentRead`	Si se establece en `true`, se utiliza una lectura consistente para la tabla especificada; en caso contrario, se utiliza una lectura consistente final. Tipo: booleano.

Errores especiales

Error	Descripción
`ProvisionedThroughputExceededException`	Se ha superado el desempeño provisionado máximo permitido.

Ejemplos

En los siguientes ejemplos se muestra una solicitud HTTP POST y su respuesta utilizando la operación BatchGetItem. Para obtener ejemplos sobre cómo usar el SDK de AWS, consulte Uso de elementos y atributos en DynamoDB.

Solicitud de ejemplo

En el ejemplo siguiente se solicitan atributos de dos tablas diferentes.

```
// This header is abbreviated.
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0
content-length: 409

{"RequestItems":
    {"comp1":
        {"Keys":
            [{"HashKeyElement":{"S":"Casey"},"RangeKeyElement":{"N":"1319509152"}},
            {"HashKeyElement":{"S":"Dave"},"RangeKeyElement":{"N":"1319509155"}},
            {"HashKeyElement":{"S":"Riley"},"RangeKeyElement":{"N":"1319509158"}}],
        "AttributesToGet":["user","status"]},
    "comp2":
        {"Keys":
            [{"HashKeyElement":{"S":"Julie"}},{"HashKeyElement":{"S":"Mingus"}}],
        "AttributesToGet":["user","friends"]}
    }
}
```

Respuesta de ejemplo

El ejemplo siguiente es la respuesta.

```
HTTP/1.1 200 OK
x-amzn-RequestId: GTPQVRM4VJS792J1UFJTKUBVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 373
Date: Fri, 02 Sep 2011 23:07:39 GMT

{"Responses":
    {"comp1":
```

```
        {"Items":
            [{"status":{"S":"online"},"user":{"S":"Casey"}},
            {"status":{"S":"working"},"user":{"S":"Riley"}},
            {"status":{"S":"running"},"user":{"S":"Dave"}}],
        "ConsumedCapacityUnits":1.5},
    "comp2":
        {"Items":
            [{"friends":{"SS":["Elisabeth", "Peter"]},"user":{"S":"Mingus"}},
            {"friends":{"SS":["Dave", "Peter"]},"user":{"S":"Julie"}}],
        "ConsumedCapacityUnits":1}
    },
    "UnprocessedKeys":{}
}
```

BatchWriteItem

> **Important**
>
> ***Esta sección se refiere a la versión 2011-12-05 del API, que está obsoleta y no debe utilizarse para nuevas aplicaciones.***
>
> Para consultar la documentación sobre la API de bajo nivel actual, consulte la Referencia de la API de Amazon DynamoDB.

Descripción

Esta operación permite colocar o eliminar varios elementos en una o varias tablas con una sola llamada.

Para cargar un elemento, puede utilizar `PutItem` y para eliminar un elemento, puede utilizar `DeleteItem`. Sin embargo, si desea cargar o eliminar grandes cantidades de datos, como, por ejemplo, cargar grandes cantidades de datos desde Amazon EMR (Amazon EMR) o migrar datos desde otra base de datos a DynamoDB, `BatchWriteItem` ofrece una alternativa eficiente.

Si utiliza lenguajes como Java, puede usar hilos para cargar los elementos en paralelo. Esto agrega complejidad a la aplicación, porque debe encargarse de los hilos. Otros lenguajes no admiten los hilos. Por ejemplo, si utiliza PHP, debe cargar o eliminar los elementos uno por uno. En ambos casos, `BatchWriteItem` proporciona una alternativa para procesar en paralelo las operaciones de colocación y eliminación, por lo que aporta la potencia de un grupo de subprocesos sin tener que aumentar la complejidad de la aplicación.

Tenga en cuenta que cada una de las operaciones individuales de colocación y eliminación que se especifican en una operación `BatchWriteItem` cuesta lo mismo en términos de unidades de capacidad consumidas. Sin embargo, puesto que `BatchWriteItem` lleva a cabo las operaciones especificadas en paralelo, se consigue una latencia menor. Cada una de las operaciones de eliminación de elementos inexistentes consumen una unidad de capacidad de escritura. Para obtener más información sobre las unidades de capacidad provisionadas, consulte Uso de tablas y datos en DynamoDB.

Cuando utilice `BatchWriteItem`, tenga en cuenta las siguientes limitaciones:

- Número máximo de operaciones en una sola solicitud: puede especificar un máximo de 25 operaciones de colocación o eliminación en total; sin embargo, el tamaño total de la solicitud no puede ser mayor que 1 MB (la carga de HTTP).
- Puede utilizar la operación `BatchWriteItem` únicamente para colocar y eliminar elementos. No la puede utilizar para actualizar los existentes.
- No es una operación atómica: las operaciones individuales especificadas en una operación `BatchWriteItem` son atómicas; sin embargo, `BatchWriteItem` en su conjunto es una operación basada en el principio "en la medida en que sea posible", no es atómica. Es decir, en una solicitud `BatchWriteItem`, algunas operaciones podrían realizarse correctamente y otras, no. Las operaciones que presentan algún error se devuelven en el campo `UnprocessedItems` en la respuesta. Algunos de estos errores podrían deberse a que se haya superado el desempeño provisionado configurado para la tabla, o a un error transitorio de la red. Puede investigar y, si lo desea, reenviar las solicitudes. Normalmente, se llama a `BatchWriteItem` en bucle, se comprueba en cada iteración si quedan elementos sin procesar y se envía una nueva solicitud `BatchWriteItem` con esos elementos que faltan.
- No se devuelven elementos: la operación `BatchWriteItem` se ha diseñado para cargar grandes cantidades de datos de forma eficiente. No proporciona la sofisticación que ofrecen `PutItem` y `DeleteItem`. Por ejemplo, `DeleteItem` es compatible con el campo `ReturnValues` del cuerpo de la solicitud para solicitar el elemento eliminado en la respuesta. En cambio, la operación `BatchWriteItem` no devuelve ningún elemento de respuesta.
- A diferencia de `PutItem` y `DeleteItem`, `BatchWriteItem` no permite especificar condiciones para solicitudes de escritura individuales de la operación.
- Los valores de los atributos no pueden ser null; los atributos de tipo cadena y binario deben tener una longitud superior a cero; y los atributos de tipo conjunto no pueden estar vacíos. Las solicitudes con valores vacíos se rechazan con la excepción `ValidationException`.

DynamoDB rechaza toda la operación de escritura por lotes si se cumple cualquiera de las siguientes condiciones:

- Si una o varias tablas especificados en la solicitud `BatchWriteItem` no existe.
- Si los atributos de clave principal especificados en un elemento de la solicitud no coinciden con el esquema de clave principal de la tabla correspondiente.
- Si intenta realizar varias operaciones con el mismo elemento en la misma solicitud `BatchWriteItem`. Por ejemplo, no puede colocar y eliminar el mismo elemento en la misma solicitud `BatchWriteItem`.
- Si el tamaño total de la solicitud supera el límite de tamaño de solicitud de 1 MB (la carga de HTTP).
- Si cualquier elemento individual de un lote supera el límite de tamaño de elemento de 64 KB.

Solicitudes

Sintaxis

```
// This header is abbreviated. For a sample of a complete header, see API de bajo nivel
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{
  "RequestItems" :  RequestItems
}

RequestItems
{
     "TableName1" :  [ Request, Request, ... ],
     "TableName2" :  [ Request, Request, ... ],
     ...
}

Request ::=
  PutRequest | DeleteRequest

PutRequest ::=
{
  "PutRequest" : {
```

```
     "Item" : {
        "Attribute-Name1" : Attribute-Value,
        "Attribute-Name2" : Attribute-Value,
         ...
     }
  }
}

DeleteRequest ::=
{
   "DeleteRequest" : {
      "Key" : PrimaryKey-Value
   }
}


PrimaryKey-Value ::= HashTypePK | HashAndRangeTypePK

HashTypePK ::=
{
   "HashKeyElement" : Attribute-Value
}

HashAndRangeTypePK
{
   "HashKeyElement" : Attribute-Value,
   "RangeKeyElement" : Attribute-Value,
}

Attribute-Value ::= String | Numeric| Binary | StringSet | NumericSet | BinarySet

Numeric ::=
{
   "N": "Number"
}

String ::=
{
   "S": "String"
}

Binary ::=
{
    "B": "Base64 encoded binary data"
```

```
}

StringSet ::=
{
   "SS": [ "String1", "String2", ... ]
}

NumberSet ::=
{
   "NS": [ "Number1", "Number2", ... ]
}

BinarySet ::=
{
   "BS": [ "Binary1", "Binary2", ... ]
}
```

En el cuerpo de la solicitud, el objeto JSON `RequestItems` describe las operaciones que desea a realizar. Las operaciones se agrupan por tablas. Puede utilizar `BatchWriteItem` para actualizar o eliminar varios elementos en varias tablas. Para cada solicitud de escritura específica, debe identificar el tipo de solicitud (`PutItem` o `DeleteItem`), seguido de información detallada sobre la operación.

- Para una solicitud `PutRequest`, se proporciona el elemento, es decir, una lista de atributos y sus valores.
- Para una solicitud `DeleteRequest`, se proporcionan el nombre y el valor de la clave principal.

Respuestas

Sintaxis

A continuación encontrará la sintaxis del cuerpo JSON devuelto en la respuesta.

```
{
  "Responses" :         ConsumedCapacityUnitsByTable
  "UnprocessedItems" :  RequestItems
}

ConsumedCapacityUnitsByTable
{
    "TableName1" : { "ConsumedCapacityUnits", : NumericValue },
```

```
    "TableName2" : { "ConsumedCapacityUnits", : NumericValue },
     ...
}

RequestItems
This syntax is identical to the one described in the JSON syntax in the request.
```

Errores especiales

No hay errores específicos de esta operación.

Ejemplos

En el siguiente ejemplo se muestra una solicitud HTTP POST y la respuesta de una operación `BatchWriteItem`. En la solicitud se especifican las siguientes operaciones en las tablas Reply y Thread:

- Colocar un elemento y eliminar un elemento de la tabla Reply
- Colocar un elemento en la tabla Thread

Para obtener ejemplos sobre cómo usar el SDK de AWS, consulte Uso de elementos y atributos en DynamoDB.

Solicitud de ejemplo

```
// This header is abbreviated. For a sample of a complete header, see API de bajo nivel
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.BatchGetItem
content-type: application/x-amz-json-1.0

{
  "RequestItems":{
    "Reply":[
      {
        "PutRequest":{
          "Item":{
            "ReplyDateTime":{
              "S":"2012-04-03T11:04:47.034Z"
            },
            "Id":{
              "S":"DynamoDB#DynamoDB Thread 5"
```

```
            }
          }
        }
      },
      {
        "DeleteRequest":{
          "Key":{
            "HashKeyElement":{
              "S":"DynamoDB#DynamoDB Thread 4"
            },
            "RangeKeyElement":{
              "S":"oops - accidental row"
            }
          }
        }
      }
    ],
    "Thread":[
      {
        "PutRequest":{
          "Item":{
            "ForumName":{
              "S":"DynamoDB"
            },
            "Subject":{
              "S":"DynamoDB Thread 5"
            }
          }
        }
      }
    ]
  }
}
```

Respuesta de ejemplo

En la respuesta del siguiente ejemplo se muestra una operación de colocación que se ha realizado correctamente en las tablas Thread y Reply, y una operación que no se ha podido realizar en la tabla Reply (por ejemplo, porque se ha aplicado una limitación controlada por haberse superado el rendimiento aprovisionado de la tabla). Observe lo siguiente en la respuesta de JSON:

- El objeto `Responses` muestra que se ha consumido una unidad de capacidad en cada tabla, `Thread` y `Reply`, debido a sendas operaciones de colocación llevadas a cabo en ellas.

- El objeto `UnprocessedItems` muestra la operación de eliminación que no se pudo realizar en la tabla `Reply`. A continuación, puede emitir una nueva llamada a `BatchWriteItem` para repetir estas solicitudes sin procesar.

```
HTTP/1.1 200 OK
x-amzn-RequestId: G8M9ANLOE5QA26AEUHJKJE0ASBVV4KQNSO5AEMVJF66Q9ASUAAJG
Content-Type: application/x-amz-json-1.0
Content-Length: 536
Date: Thu, 05 Apr 2012 18:22:09 GMT

{
   "Responses":{
      "Thread":{
         "ConsumedCapacityUnits":1.0
      },
      "Reply":{
         "ConsumedCapacityUnits":1.0
      }
   },
   "UnprocessedItems":{
      "Reply":[
         {
            "DeleteRequest":{
               "Key":{
                  "HashKeyElement":{
                     "S":"DynamoDB#DynamoDB Thread 4"
                  },
                  "RangeKeyElement":{
                     "S":"oops - accidental row"
                  }
               }
            }
         }
      ]
   }
}
```

CreateTable

> **Important**
>
> ***Esta sección se refiere a la versión 2011-12-05 del API, que está obsoleta y no debe utilizarse para nuevas aplicaciones.***
>
> Para consultar la documentación sobre la API de bajo nivel actual, consulte la Referencia de la API de Amazon DynamoDB.

Descripción

La operación `CreateTable` agrega una nueva tabla a la cuenta.

El nombre de la tabla debe ser único para la cuenta de AWS que emite la solicitud y la región de AWS que recibe la solicitud a las que están asociadas (por ejemplo, dynamodb.us-west-2.amazonaws.com). Cada punto de enlace de DynamoDB es totalmente independiente. Por ejemplo, si tiene dos tablas llamadas “MyTable”, una en dynamodb.us-west-2.amazonaws.com y otra en dynamodb.us-west-1.amazonaws.com, son completamente independientes y no comparten ningún dato.

La operación `CreateTable` desencadena un flujo de trabajo asincrónico para comenzar a crear la tabla. DynamoDB devuelve inmediatamente el estado de la tabla (`CREATING`) hasta que la tabla adquiere el estado `ACTIVE`. Una vez que la tabla tiene el estado `ACTIVE`, puede llevar a cabo operaciones del plano de datos.

Use la operación DescribeTables para comprobar el estado de la tabla.

Solicitudes

Sintaxis

```
// This header is abbreviated.
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.CreateTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
```

```
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10}
}
```

Nombre	Descripción	Obligatorio
`TableName`	Nombre de la tabla que se va a crear. Los caracteres permitidos son a-z, A-Z, 0-9, "_" (guion bajo), "-" (guion) y "." (punto). Los nombres pueden tener de 3 a 255 caracteres de longitud. Tipo: cadena	Sí
`KeySchema`	Estructura de la clave principal (simple o compuesta) de la tabla. Se requiere un par de nombre-valor de `HashKeyElement` , pero el par de nombre-valor de `RangeKeyElement` es opcional (solo es obligatorio para las claves principales compuestas). Para obtener más información sobre claves principales, consulte Clave principal. Los nombres de elemento de clave principal pueden tener entre 1 y 255 caracteres y no tienen restricciones de caracteres. Los valores posibles de AttributeType son	Sí

Nombre	Descripción	Obligatorio
	"S" (cadena), "N" (numérico) o "B" (binario). Tipo: Map, mapa de `HashKeyElement` , o bien de `HashKeyElement` y de `RangeKeyElement` si se trata de una clave principal compuesta.	
`ProvisionedThroughput`	Nuevo rendimiento de la tabla especificada, que se compone de los valores de `ReadCapacityUnits` y `WriteCapacityUnits` . Para obtener más información, consulte Modo de capacidad aprovisionada de DynamoDB. **Note** Para obtener los valores mínimo y máximo actuales, consulte Cuotas en Amazon DynamoDB. Tipo: matriz	Sí

Nombre	Descripción	Obligatorio
`ProvisionedThroughput : ReadCapacityUnits`	Establece el número mínimo de `ReadCapacityUnits` consistentes consumidas por segundo para la tabla especificada antes de que DynamoDB equilibre la carga con otras operaciones. Las operaciones de lectura consistente final requieren menos esfuerzo que una operación de lectura consistente; por lo tanto, un ajuste de 50 unidades `ReadCapacityUnits` consistentes por segundo proporciona 100 unidades de capacidad `ReadCapacityUnits` consistentes finales por segundo. Tipo: Number	Sí
`ProvisionedThroughput : WriteCapacityUnits`	Establece el número mínimo de `WriteCapacityUnits` consumidas por segundo para la tabla especificada antes de que DynamoDB equilibre la carga con otras operaciones. Tipo: Number	Sí

Respuestas

Sintaxis

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.310506263362E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableStatus":"CREATING"
    }
}
```

Nombre	Descripción
`TableDescription`	Contenedor de propiedades de la tabla.
`CreationDateTime`	Fecha en que se creó la tabla en formato de tiempo UNIX. Tipo: Number
`KeySchema`	Estructura de la clave principal (simple o compuesta) de la tabla. Se requiere un par de nombre-valor de `HashKeyElement` , pero el par de nombre-valor de `RangeKeyElement` es opcional (solo es obligatorio para las claves principales compuestas). Para obtener más información sobre claves principales, consulte Clave principal.

Nombre	Descripción
	Tipo: Map, mapa de `HashKeyElement` , o bien de `HashKeyElement` y de `RangeKeyElement` si se trata de una clave principal compuesta.
`ProvisionedThroughput`	Rendimiento de la tabla especificada, que se compone de los valores de `ReadCapacityUnits` y `WriteCapacityUnits` . Consulte Modo de capacidad aprovisionada de DynamoDB. Tipo: matriz
`ProvisionedThroughput:ReadCapacityUnits`	El número mínimo de `ReadCapacityUnits` consumidas por segundo antes de que DynamoDB equilibre la carga con otras operaciones. Tipo: Number
`ProvisionedThroughput:WriteCapacityUnits`	El número mínimo de `ReadCapacityUnits` consumidas por segundo antes de que `WriteCapacityUnits` equilibre la carga con otras operaciones. Tipo: Number
`TableName`	Nombre de la tabla creada. Tipo: cadena

Nombre	Descripción
`TableStatus`	Estado actual de la tabla (`CREATING`). Una vez que la tabla adquiere el estado `ACTIVE`, puede colocar datos en ella. Use el API DescribeTables para comprobar el estado de la tabla. Tipo: cadena

Errores especiales

Error	Descripción
`ResourceInUseException`	Se intentó volver a crear una tabla que ya existía.
`LimitExceededException`	El número de solicitudes simultáneas para la tabla (número acumulado de tablas que se encuentren en los estados `CREATING`, `DELETING` o `UPDATING`) supera el máximo permitido. **Note** Para obtener los valores mínimo y máximo actuales, consulte Cuotas en Amazon DynamoDB. .

Ejemplos

En el siguiente ejemplo se crea una tabla con una clave principal compuesta que contiene una cadena y un número. Para obtener ejemplos sobre cómo usar el SDK de AWS, consulte Uso de tablas y datos en DynamoDB.

Solicitud de ejemplo

```
// This header is abbreviated.
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.CreateTable
content-type: application/x-amz-json-1.0


{"TableName":"comp-table",
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10}
}
```

Respuesta de ejemplo

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"TableDescription":
    {"CreationDateTime":1.310506263362E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":10},
    "TableName":"comp-table",
    "TableStatus":"CREATING"
    }
}
```

Acciones relacionadas

- DescribeTables
- DeleteTable

DeleteItem

> **Important**
>
> ***Esta sección se refiere a la versión 2011-12-05 del API, que está obsoleta y no debe utilizarse para nuevas aplicaciones.***
> Para consultar la documentación sobre la API de bajo nivel actual, consulte la Referencia de la API de Amazon DynamoDB.

Descripción

Elimina un solo elemento de una tabla por su clave principal Puede realizar una operación de eliminación condicional que elimina el elemento si existe o si tiene un valor de atributo esperado.

> **Note**
>
> Si especifica `DeleteItem` sin atributos ni valores, se eliminan todos los atributos del elemento.
> A menos que especifique condiciones, `DeleteItem` es una operación idempotente; aunque se ejecute varias veces en el mismo elemento o atributo no se obtiene una respuesta de error.
> Las eliminaciones condicionales solo son útiles para eliminar elementos y atributos cuando se cumplen determinadas condiciones. Si se cumplen las condiciones, DynamoDB lleva a cabo la eliminación. De lo contrario, el elemento no se elimina.
> Puede llevar a cabo la comprobación condicional esperada en un atributo por operación.

Solicitudes

Sintaxis

```
// This header is abbreviated.
```

```
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Key":
        {"HashKeyElement":{"S":"AttributeValue1"},"RangeKeyElement":
{"N":"AttributeValue2"}},
    "Expected":{"AttributeName3":{"Value":{"S":"AttributeValue3"}}},
    "ReturnValues":"ALL_OLD"}
}
```

Nombre	Descripción	Obligatorio
`TableName`	Nombre de la tabla que contiene el elemento que se va a eliminar. Tipo: cadena	Sí
`Key`	Clave principal que define el elemento. Para obtener más información sobre claves principales, consulte Clave principal. Tipo: Map, mapa de `HashKeyElement` a su valor y `RangeKeyElement` a su valor.	Sí
`Expected`	Designa un atributo para una eliminación condicional. El parámetro `Expected` le permite proporcionar un nombre de atributo e indicar si DynamoDB debe verificar o no si el atributo tiene un valor	No

Nombre	Descripción	Obligatorio
	determinado antes de eliminarlo. Tipo: Map, mapa de nombres de atributos.	
`Expected:AttributeName`	Nombre del atributo para la operación Put condicional. Tipo: cadena	No

Nombre	Descripción	Obligatorio

`Expected:AttributeName: ExpectedAttributeValue`	Use este parámetro para especificar si ya existe un valor del par de nombre-valor del atributo.
	
	En la siguiente notación JSON, se elimina el elemento si todavía no existe el atributo "Color" para ese elemento:
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	En la siguiente notación JSON se comprueba si el atributo denominado "Color" tiene el valor "Yellow" antes de eliminar el elemento:
	
	`"Expected" : {"Color":{"Exists":true},{"Value": {"S":"Yellow"}}}`
	
	De forma predeterminada, si utiliza el parámetro `Expected` y le proporciona un `Value`, DynamoDB da por hecho que el atributo existe y que posee un valor que hay que sustituir. Por lo tanto, no es preciso especificar `{"Exists":true}` , porque se considera implícito. Puede reducir la solicitud a:
		No

Nombre	Descripción	Obligatorio
	`"Expected" : {"Color":{"Value":{"S":"Yellow"}}}` **Note** Si especifica `{"Exists":true}` sin un valor de atributo que verificar, DynamoDB devuelve un error.	
`ReturnValues`	Use este parámetro si desea obtener los pares de nombre-valor de los atributos antes de eliminarlos. Los valores posibles de los parámetros son `NONE` (predeterminado) o `ALL_OLD`. Si se especific a `ALL_OLD`, se devuelve el contenido del elemento anterior. Si este parámetro no se proporciona o si su valor es `NONE`, no se devuelve nada. Tipo: cadena	No

Respuestas

Sintaxis

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
```

```
content-length: 353
Date: Tue, 12 Jul 2011 21:31:03 GMT

{"Attributes":
    {"AttributeName3":{"SS":["AttributeValue3","AttributeValue4","AttributeValue5"]},
    "AttributeName2":{"S":"AttributeValue2"},
    "AttributeName1":{"N":"AttributeValue1"}
    },
"ConsumedCapacityUnits":1
}
```

Nombre	Descripción
`Attributes`	Si el parámetro `ReturnValues` se suministra como `ALL_OLD` en la solicitud, DynamoDB devuelve una matriz de pares de nombre-valor del atributo (es decir, el elemento eliminado). De lo contrario, la respuesta contiene un conjunto vacío. Tipo: Array; matriz de pares de nombre-valor del atributo.
`ConsumedCapacityUnits`	Cantidad de unidades de capacidad de escritura consumidas por la operación. Este valor muestra el número aplicado al rendimiento aprovisionado. Cada solicitud de eliminación de elementos inexistentes consume una unidad de capacidad de escritura. Para obtener más información, consulte Modo de capacidad aprovisionada de DynamoDB. Tipo: Number

Errores especiales

Error	Descripción
`ConditionalCheckFailedException`	Error en la verificación condicional. No se encontró un valor de atributo esperado.

Ejemplos

Solicitud de ejemplo

```
// This header is abbreviated.
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteItem
content-type: application/x-amz-json-1.0

{"TableName":"comp-table",
    "Key":
        {"HashKeyElement":{"S":"Mingus"},"RangeKeyElement":{"N":"200"}},
    "Expected":
        {"status":{"Value":{"S":"shopping"}}},
    "ReturnValues":"ALL_OLD"
}
```

Respuesta de ejemplo

```
HTTP/1.1 200 OK
x-amzn-RequestId: U9809LI6BBFJA5N2R0TB0P017JVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 353
Date: Tue, 12 Jul 2011 22:31:23 GMT

{"Attributes":
    {"friends":{"SS":["Dooley","Ben","Daisy"]},
    "status":{"S":"shopping"},
    "time":{"N":"200"},
    "user":{"S":"Mingus"}
    },
"ConsumedCapacityUnits":1
```

```
}
```

Acciones relacionadas

- PutItem

DeleteTable

> **Important**
>
> ***Esta sección se refiere a la versión 2011-12-05 del API, que está obsoleta y no debe utilizarse para nuevas aplicaciones.***
>
> Para consultar la documentación sobre la API de bajo nivel actual, consulte la Referencia de la API de Amazon DynamoDB.

Descripción

La operación `DeleteTable` elimina una tabla y todos sus elementos. Después de una solicitud `DeleteTable`, la tabla especificada se encuentra en estado `DELETING` hasta que DynamoDB completa la eliminación. Si la tabla está en estado `ACTIVE`, puede eliminarla. Si una tabla se encuentra en los estados `CREATING` o `UPDATING`, entonces DynamoDB devuelve un error `ResourceInUseException`. Si la tabla especificada no existe, DynamoDB devuelve una `ResourceNotFoundException`. Si la tabla ya se encuentra en el estado `DELETING`, no se devuelve ningún error.

> **Note**
>
> DynamoDB podría continuar aceptando solicitudes de operaciones con el plano de datos, tales como `GetItem` y `PutItem`, en una tabla que se encuentre en estado `DELETING` hasta que se haya completado la eliminación de la tabla.

Las tablas son únicas entre las asociadas a la cuenta de AWS que emite la solicitud y la región de AWS que recibe la solicitud a las que están asociadas (por ejemplo, dynamodb.us-west-1.amazonaws.com). Cada punto de enlace de DynamoDB es totalmente independiente. Por ejemplo, si tiene dos tablas llamadas “MyTable”, una en dynamodb.us-west-2.amazonaws.com y

otra en dynamodb.us-west-1.amazonaws.com, son completamente independientes y no comparten ningún dato; al eliminar una no se elimina la otra.

Use la operación DescribeTables para comprobar el estado de la tabla.

Solicitudes

Sintaxis

```
// This header is abbreviated.
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

Nombre	Descripción	Obligatorio
`TableName`	Nombre de la tabla que se va a eliminar. Tipo: cadena	Sí

Respuestas

Sintaxis

```
HTTP/1.1 200 OK
x-amzn-RequestId: 4HONCKIVH1BFUDQ1U68CTG3N27VV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 311
Date: Sun, 14 Aug 2011 22:56:22 GMT

{"TableDescription":
    {"CreationDateTime":1.313362508446E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"ReadCapacityUnits":10,"WriteCapacityUnits":10},
    "TableName":"Table1",
```

```
    "TableStatus":"DELETING"
    }
}
```

Nombre	Descripción
`TableDescription`	Contenedor de propiedades de la tabla.
`CreationDateTime`	Fecha de creación de la tabla. Tipo: Number
`KeySchema`	Estructura de la clave principal (simple o compuesta) de la tabla. Se requiere un par de nombre-valor de `HashKeyElement` , pero el par de nombre-valor de `RangeKeyElement` es opcional (solo es obligatorio para las claves principales compuestas). Para obtener más información sobre claves principales, consulte Clave principal. Tipo: Map, mapa de `HashKeyElement` , o bien de `HashKeyElement` y de `RangeKeyElement` si se trata de una clave principal compuesta.
`ProvisionedThroughput`	Rendimiento de la tabla especificada, que se compone de los valores de `ReadCapacityUnits` y `WriteCapacityUnits` . Consulte Modo de capacidad aprovisionada de DynamoDB.
`ProvisionedThroughput` : `ReadCapacityUnits`	Cantidad mínima de `ReadCapacityUnits` consumidas por segundo para la tabla especificada antes de que DynamoDB equilibre la carga con otras operaciones. Tipo: Number

Nombre	Descripción
`ProvisionedThroughput:WriteCapacityUnits`	Cantidad mínima de `WriteCapacityUnits` consumidas por segundo para la tabla especificada antes de que DynamoDB equilibre la carga con otras operaciones. Tipo: Number
`TableName`	Nombre de la tabla eliminada. Tipo: cadena
`TableStatus`	Estado actual de la tabla (`DELETING`). Una vez que se elimina la tabla, las solicitudes posteriores de la tabla devuelven `resource not found`. Use la operación DescribeTables para comprobar el estado de la tabla. Tipo: cadena

Errores especiales

Error	Descripción
`ResourceInUseException`	La tabla se encuentra en el estado `CREATING` o `UPDATING` y no se puede eliminar.

Ejemplos

Solicitud de ejemplo

```
// This header is abbreviated. For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DeleteTable
```

```
content-type: application/x-amz-json-1.0
content-length: 40

{"TableName":"favorite-movies-table"}
```

Respuesta de ejemplo

```
HTTP/1.1 200 OK
x-amzn-RequestId: 4HONCKIVH1BFUDQ1U68CTG3N27VV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 160
Date: Sun, 14 Aug 2011 17:20:03 GMT

{"TableDescription":
    {"CreationDateTime":1.313362508446E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"name","AttributeType":"S"}},
    "TableName":"favorite-movies-table",
    "TableStatus":"DELETING"
}
```

Acciones relacionadas

- CreateTable
- DescribeTables

DescribeTables

> **Important**
>
> ***Esta sección se refiere a la versión 2011-12-05 del API, que está obsoleta y no debe utilizarse para nuevas aplicaciones.***
>
> Para consultar la documentación sobre la API de bajo nivel actual, consulte la Referencia de la API de Amazon DynamoDB.

Descripción

Devuelve información sobre la tabla, incluyendo su estado actual, el esquema de clave principal y el momento de creación. Los resultados de DescribeTable son eventualmente consistentes. Si utiliza

DescribeTable demasiado pronto en el proceso de creación de una tabla, DynamoDB devuelve una `ResourceNotFoundException`. Si utiliza DescribeTable demasiado pronto en el proceso de actualización de una tabla, los nuevos valores podrían no estar disponibles inmediatamente.

Solicitudes

Sintaxis

```
// This header is abbreviated.
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1"}
```

Nombre	Descripción	Obligatorio
`TableName`	Nombre de la tabla que se describe. Tipo: cadena	Sí

Respuestas

Sintaxis

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
Content-Length: 543


{"Table":
    {"CreationDateTime":1.309988345372E9,
    ItemCount:1,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeName1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeName2","AttributeType":"N"}},
    "ProvisionedThroughput":{"LastIncreaseDateTime": Date, "LastDecreaseDateTime":
 Date, "ReadCapacityUnits":10,"WriteCapacityUnits":10},
```

```
    "TableName":"Table1",
    "TableSizeBytes":1,
    "TableStatus":"ACTIVE"
    }
}
```

Nombre	Descripción
`Table`	Contenedor de la tabla que se describe. Tipo: cadena
`CreationDateTime`	Fecha en que se creó la tabla en formato de tiempo UNIX.
`ItemCount`	Cantidad de elementos de la tabla especific ada. DynamoDB actualiza este valor aproximad amente cada seis horas. Los cambios recientes podrían no reflejarse en este valor. Tipo: Number
`KeySchema`	Estructura de la clave principal (simple o compuesta) de la tabla. Se requiere un par de nombre-valor de `HashKeyElement` , pero el par de nombre-valor de `RangeKeyElement` es opcional (solo es obligatorio para las claves principales compuestas). El tamaño máximo de la clave hash es de 2048 bytes. El tamaño máximo de la clave de rango es de 1024 bytes. Ambos límites se aplican por separado (es decir, puede disponer de una clave combinada de rango y hash de 2048+1024). Para obtener más información sobre claves principales, consulte Clave principal.
`ProvisionedThroughput`	Rendimiento de la tabla especificada, que consta de los valores de `LastIncre aseDateTime` (si procede), `LastDecre`

Nombre	Descripción
	`aseDateTime` (si procede), `ReadCapac ityUnits` y `WriteCapacityUnits`. Si el rendimiento de la tabla no se ha aumentado ni disminuido, DynamoDB no devuelve valores para estos elementos. Consulte Modo de capacidad aprovisionada de DynamoDB. Tipo: matriz
`TableName`	Nombre de la tabla solicitada. Tipo: cadena
`TableSizeBytes`	Tamaño total de la tabla especificada, en bytes. DynamoDB actualiza este valor aproximad amente cada seis horas. Los cambios recientes podrían no reflejarse en este valor. Tipo: Number
`TableStatus`	Estado actual de la tabla (`CREATING`, `ACTIVE`, `DELETING` o `UPDATING`). Una vez que la tabla adquiere el estado `ACTIVE`, puede agregarle datos.

Errores especiales

No hay errores específicos de esta operación.

Ejemplos

En los siguientes ejemplos se muestran una solicitud y una respuesta HTTP POST aplicando la operación DescribeTable a una tabla denominada "comp-table". La tabla tiene una clave principal compuesta.

Solicitud de muestra

```
// This header is abbreviated.
```

```
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.DescribeTable
content-type: application/x-amz-json-1.0

{"TableName":"users"}
```

Respuesta de ejemplo

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 543

{"Table":
    {"CreationDateTime":1.309988345372E9,
    "ItemCount":23,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":{"LastIncreaseDateTime": 1.309988345384E9,
 "ReadCapacityUnits":10,"WriteCapacityUnits":10},
    "TableName":"users",
    "TableSizeBytes":949,
    "TableStatus":"ACTIVE"
    }
}
```

Acciones relacionadas

- CreateTable
- DeleteTable
- ListTables

GetItem

> **Important**
>
> ***Esta sección se refiere a la versión 2011-12-05 del API, que está obsoleta y no debe utilizarse para nuevas aplicaciones.***

> Para consultar la documentación sobre la API de bajo nivel actual, consulte la Referencia de la API de Amazon DynamoDB.

Descripción

La operación `GetItem` devuelve un conjunto de `Attributes` de un elemento que coincida con la clave principal. Si no hay ningún elemento que coincida, `GetItem` no devuelve ningún dato.

De forma predeterminada, la operación `GetItem` proporciona una lectura eventualmente consistente. Si las lecturas coherentes posteriores no son aceptables en su aplicación, use `ConsistentRead`. Aunque esta operación puede tardar más que una lectura normal, siempre devuelve el último valor actualizado. Para obtener más información, consulte Coherencia de lectura de DynamoDB.

Solicitudes

Sintaxis

```
// This header is abbreviated.
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
  "Key":
  {"HashKeyElement": {"S":"AttributeValue1"},
  "RangeKeyElement": {"N":"AttributeValue2"}
 },
 "AttributesToGet":["AttributeName3","AttributeName4"],
 "ConsistentRead":Boolean
}
```

Nombre	Descripción	Obligatorio
`TableName`	Nombre de la tabla que contiene el elemento solicitado.	Sí

Nombre	Descripción	Obligatorio
	Tipo: cadena	
`Key`	Valores de clave principal que definen el elemento. Para obtener más informaci ón sobre claves principales, consulte Clave principal. Tipo: Map, mapa de `HashKeyElement` a su valor y `RangeKeyElement` a su valor.	Sí
`AttributesToGet`	Matriz de nombres de atributo. Si no se especifican sus nombres, se devuelven todos los atributos. Si algún atributo no se encuentra, no aparecerá en los resultados. Tipo: matriz	No
`ConsistentRead`	Si se establece en `true`, se emite una lectura consisten te; en caso contrario, se utiliza una lectura consistente final. Tipo: Booleano	No

Respuestas

Sintaxis

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 144
```

```
{"Item":{
 "AttributeName3":{"S":"AttributeValue3"},
 "AttributeName4":{"N":"AttributeValue4"},
 "AttributeName5":{"B":"dmFsdWU="}
 },
"ConsumedCapacityUnits": 0.5
}
```

Nombre	Descripción
`Item`	Contiene los atributos solicitados. Tipo: Map, mapa de pares de nombre-valor del atributo.
`ConsumedCapacityUnits`	Cantidad de unidades de capacidad de lectura consumidas por la operación. Este valor muestra el número aplicado al rendimiento aprovisionado. Las solicitudes de elementos inexistentes consumen las unidades de capacidad de lectura mínimas, según el tipo de lectura. Para obtener más información, consulte Modo de capacidad aprovisionada de DynamoDB. Tipo: Number

Errores especiales

No hay errores específicos de esta operación.

Ejemplos

Para obtener ejemplos sobre cómo usar el SDK de AWS, consulte Uso de elementos y atributos en DynamoDB.

Solicitud de ejemplo

```
// This header is abbreviated.
```

```
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.GetItem
content-type: application/x-amz-json-1.0

{"TableName":"comptable",
 "Key":
  {"HashKeyElement":{"S":"Julie"},
  "RangeKeyElement":{"N":"1307654345"}},
 "AttributesToGet":["status","friends"],
 "ConsistentRead":true
}
```

Respuesta de ejemplo

Observe que el valor de ConsumedCapacityUnits es 1, ya que el parámetro opcional `ConsistentRead` se ha establecido en `true`. Si `ConsistentRead` se establece en `false` (o no se especifica) para la misma solicitud, la respuesta presentará consistencia final, en cuyo caso el valor de ConsumedCapacityUnits sería 0,5.

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 72

{"Item":
 {"friends":{"SS":["Lynda, Aaron"]},
 "status":{"S":"online"}
 },
"ConsumedCapacityUnits": 1
}
```

ListTables

> **Important**
>
> ***Esta sección se refiere a la versión 2011-12-05 del API, que está obsoleta y no debe utilizarse para nuevas aplicaciones.***
>
> Para consultar la documentación sobre la API de bajo nivel actual, consulte la Referencia de la API de Amazon DynamoDB.

Descripción

Muestra una matriz de todas las tablas asociadas con la cuenta y el punto de enlace actuales.

Cada punto de enlace de DynamoDB es totalmente independiente. Por ejemplo, si tiene dos tablas llamadas “MyTable”, una en dynamodb.us-west-2.amazonaws.com y otra en dynamodb.us-east-1.amazonaws.com, son completamente independientes y no comparten ningún dato. La operación ListTables devuelve todos los nombres de las tablas asociadas con la cuenta que realiza la solicitud para el punto de enlace que recibe la solicitud.

Solicitudes

Sintaxis

```
// This header is abbreviated.
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"Table1","Limit":3}
```

De forma predeterminada, la operación ListTables solicita todos los nombres de las tablas asociadas con la cuenta que realiza la solicitud para el punto de enlace que recibe la solicitud.

Nombre	Descripción	Obligatorio
`Limit`	Número máximo de nombres de tabla que se devolverán. Tipo: entero	No
`ExclusiveStartTableName`	Nombre de la primera tabla de la lista. Si ya se ha ejecutado una operación ListTables y se ha recibido un valor `LastEvaluatedTableName` en la respuesta, use ese valor aquí para continuar la lista.	No

Nombre	Descripción	Obligatorio
	Tipo: cadena	

Respuestas

Sintaxis

```
HTTP/1.1 200 OK
x-amzn-RequestId: S1LEK2DPQP8OJNHVHL8OU2M7KRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 81
Date: Fri, 21 Oct 2011 20:35:38 GMT

{"TableNames":["Table1","Table2","Table3"], "LastEvaluatedTableName":"Table3"}
```

Nombre	Descripción
`TableNames`	Nombres de las tablas asociadas con la cuenta actual en el punto de enlace actual. Tipo: matriz
`LastEvaluatedTableName`	Nombre de la última tabla de la lista actual, pero solo si no se han devuelto algunas tablas de la cuenta y el punto de enlace. Este valor no existe en una respuesta si ya se han devuelto los nombres de todas las tablas. Use este valor en `ExclusiveStartTableName` en una nueva solicitud para continuar la lista hasta que se hayan devuelto los nombres de todas las tablas. Tipo: cadena

Errores especiales

No hay errores específicos de esta operación.

Ejemplos

En los ejemplos siguientes se muestra una solicitud HTTP POST y su respuesta utilizando la operación ListTables.

Solicitud de ejemplo

```
// This header is abbreviated.
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.ListTables
content-type: application/x-amz-json-1.0

{"ExclusiveStartTableName":"comp2","Limit":3}
```

Respuesta de ejemplo

```
HTTP/1.1 200 OK
x-amzn-RequestId: S1LEK2DPQP8OJNHVHL8OU2M7KRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 81
Date: Fri, 21 Oct 2011 20:35:38 GMT

{"LastEvaluatedTableName":"comp5","TableNames":["comp3","comp4","comp5"]}
```

Acciones relacionadas

- DescribeTables
- CreateTable
- DeleteTable

PutItem

> **Important**
>
> ***Esta sección se refiere a la versión 2011-12-05 del API, que está obsoleta y no debe utilizarse para nuevas aplicaciones.***
>
> Para consultar la documentación sobre la API de bajo nivel actual, consulte la Referencia de la API de Amazon DynamoDB.

Descripción

Crea un elemento nuevo o sustituye uno antiguo por uno nuevo (incluyendo todos los atributos). Si ya existe un elemento con la misma clave principal en la tabla especificada, el nuevo elemento sustituye por completo al existente. Puede realizar una colocación condicional (insertar un nuevo elemento solamente si no existe otro con la clave principal especificada) o bien sustituir un elemento existente si sus atributos tienen determinados valores.

Los valores de los atributos no pueden ser null; los atributos de tipo cadena y binario deben tener una longitud superior a cero; y los atributos de tipo conjunto no pueden estar vacíos. Las solicitudes con valores vacíos se rechazan con la excepción `ValidationException`.

> **Note**
>
> Para garantizar que un nuevo elemento no sustituya un elemento existente, utilice una operación de colocación condicional con `Exists` establecido en `false` para el o los atributos de clave principal.

Para obtener más información acerca del uso de `PutItem`, consulte Uso de elementos y atributos en DynamoDB.

Solicitudes

Sintaxis

```
// This header is abbreviated.
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.PutItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Item":{
        "AttributeName1":{"S":"AttributeValue1"},
        "AttributeName2":{"N":"AttributeValue2"},
        "AttributeName5":{"B":"dmFsdWU="}
    },
    "Expected":{"AttributeName3":{"Value": {"S":"AttributeValue"}, "Exists":Boolean}},
    "ReturnValues":"ReturnValuesConstant"}
```

Nombre	Descripción	Obligatorio
`TableName`	Nombre de la tabla que contendrá el elemento. Tipo: cadena	Sí
`Item`	Mapa de los atributos del elemento, que debe incluir los valores de clave principal que definen el elemento. Se pueden indicar otros pares de nombre-valor de los atributos del elemento. Para obtener más información sobre claves principales, consulte Clave principal. Tipo: Map, mapa de los nombres de atributos a los valores de atributos.	Sí
`Expected`	Designa un atributo para una colocación condicional. El parámetro `Expected` le permite proporcionar un nombre de atributo e indicar si DynamoDB debe verificar si el valor del atributo ya existe; o bien si el valor del atributo existe y tiene un valor determinado antes de cambiarlo. Tipo: Map, mapa de nombres de un atributo a valores de	No

Nombre	Descripción	Obligatorio
	un atributo e indicación de si existe.	
`Expected:AttributeName`	Nombre del atributo para la operación Put condicional. Tipo: cadena	No

Nombre	Descripción	Obligatorio

`Expected:AttributeName: ExpectedAttributeValue`	Use este parámetro para especificar si ya existe un valor del par de nombre-valor del atributo.
	
	En la siguiente notación JSON, se sustituye el elemento si todavía no existe el atributo "Color" para ese elemento:
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	En la siguiente notación JSON se comprueba si el atributo denominado "Color" tiene el valor "Yellow" antes de sustituir el elemento:
	
	`"Expected" : {"Color":{"Exists":true,{"Value":{"S":"Yellow"}}}`
	
	De forma predeterminada, si utiliza el parámetro `Expected` y le proporciona un `Value`, DynamoDB da por hecho que el atributo existe y que posee un valor que hay que sustituir. Por lo tanto, no es preciso especificar `{"Exists":true}` , porque
		No

Nombre	Descripción	Obligatorio
	se considera implícito. Puede reducir la solicitud a: `"Expected" : {"Color":{"Value":{"S":"Yellow"}}}` **Note** Si especifica `{"Exists":true}` sin un valor de atributo que verificar, DynamoDB devuelve un error.	
`ReturnValues`	Use este parámetro si desea obtener los pares de nombre-valor de los atributos antes de actualizarlos mediante la solicitud `PutItem`. Los valores posibles de los parámetros son `NONE` (predeterminado) o `ALL_OLD`. Si se especifica `ALL_OLD` y `PutItem` ha sobrescri to un par de nombre-valor del atributo, se devuelve el contenido del elemento anterior. Si este parámetro no se proporciona o si su valor es `NONE`, no se devuelve nada. Tipo: cadena	No

Respuestas

Sintaxis

En el ejemplo de sintaxis siguiente se supone que la solicitud ha especificado un parámetro `ReturnValues` de `ALL_OLD`; en caso contrario, la respuesta solo contiene la entrada `ConsumedCapacityUnits`.

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 85

{"Attributes":
 {"AttributeName3":{"S":"AttributeValue3"},
 "AttributeName2":{"SS":"AttributeValue2"},
 "AttributeName1":{"SS":"AttributeValue1"},
 },
"ConsumedCapacityUnits":1
}
```

Nombre	Descripción
`Attributes`	Valores de los atributos antes de la operación de colocación, pero solamente si el parámetro `ReturnValues` se especifica como `ALL_OLD` en la solicitud. Tipo: Map, mapa de pares de nombre-valor del atributo.
`ConsumedCapacityUnits`	Cantidad de unidades de capacidad de escritura consumidas por la operación. Este valor muestra el número aplicado al rendimien to aprovisionado. Para obtener más informaci ón, consulte Modo de capacidad aprovisionada de DynamoDB. Tipo: Number

Errores especiales

Error	Descripción
`ConditionalCheckFailedException`	Error en la verificación condicional. No se encontró un valor de atributo esperado.
`ResourceNotFoundException`	No se encontró el elemento o atributo especificado.

Ejemplos

Para obtener ejemplos sobre cómo usar el SDK de AWS, consulte Uso de elementos y atributos en DynamoDB.

Solicitud de ejemplo

```
// This header is abbreviated. For a sample of a complete header, see API de bajo nivel
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.PutItem
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
 "Item":
  {"time":{"N":"300"},
  "feeling":{"S":"not surprised"},
  "user":{"S":"Riley"}
  },
 "Expected":
  {"feeling":{"Value":{"S":"surprised"},"Exists":true}}
 "ReturnValues":"ALL_OLD"
}
```

Respuesta de ejemplo

```
HTTP/1.1 200
x-amzn-RequestId: 8952fa74-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 84
```

```
{"Attributes":
 {"feeling":{"S":"surprised"},
 "time":{"N":"300"},
 "user":{"S":"Riley"}},
"ConsumedCapacityUnits":1
}
```

Acciones relacionadas

- UpdateItem
- DeleteItem
- GetItem
- BatchGetItem

Consultar

> **Important**
>
> ***Esta sección se refiere a la versión 2011-12-05 del API, que está obsoleta y no debe utilizarse para nuevas aplicaciones.***
>
> Para consultar la documentación sobre la API de bajo nivel actual, consulte la Referencia de la API de Amazon DynamoDB.

Descripción

Una operación `Query` obtiene los valores de uno o varios elementos y sus atributos mediante su clave principal (`Query` solo está disponible para tablas con clave principal compuesta por una clave hash y una clave de rango). Debe proporcionar un `HashKeyValue` específico y puede delimitar el alcance de la consulta mediante operadores de comparación en el `RangeKeyValue` de la clave principal. Utilice el parámetro `ScanIndexForward` para obtener los resultados por orden ascendente o descendente según su clave de rango.

Las consultas que no devuelven resultados consumen las unidades de capacidad de lectura mínimas, según el tipo de lectura.

> **Note**
>
> Si la cantidad total de elementos que cumplen los parámetros de la consulta supera el límite de 1 MB, la consulta se detiene y se devuelven los resultados al usuario; en este caso, se facilita un `LastEvaluatedKey` para que pueda continuar la consulta en una operación posterior. A diferencia de una operación de análisis, una operación de consulta nunca devuelve un conjunto de resultados vacío y un valor `LastEvaluatedKey`. El valor de `LastEvaluatedKey` solamente se proporciona si los resultados superan 1 MB o si se ha utilizado el parámetro `Limit`.
>
> El resultado se puede establecer para una lectura consistente con el parámetro `ConsistentRead`.

Solicitudes

Sintaxis

```
// This header is abbreviated.
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
 "Limit":2,
 "ConsistentRead":true,
 "HashKeyValue":{"S":"AttributeValue1":},
 "RangeKeyCondition": {"AttributeValueList":
[{"N":"AttributeValue2"}],"ComparisonOperator":"GT"}
 "ScanIndexForward":true,
 "ExclusiveStartKey":{
  "HashKeyElement":{"S":"AttributeName1"},
  "RangeKeyElement":{"N":"AttributeName2"}
 },
    "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
}
```

Nombre	Descripción	Obligatorio
`TableName`	Nombre de la tabla que contiene los elementos solicitados. Tipo: cadena	Sí
`AttributesToGet`	Matriz de nombres de atributo. Si no se especifican sus nombres, se devuelven todos los atributos. Si algún atributo no se encuentra, no aparecerá en los resultados. Tipo: matriz	No
`Limit`	Cantidad máxima de elementos que se devolverá n, que no es necesariamente el número de elementos coincidentes. Si DynamoDB alcanza el límite de la cantidad de elementos mientras consulta la tabla, detiene la consulta y devuelve los valores coincidentes hasta ese punto, junto con un `LastEvaluatedKey` , que puede aplicarse en una operación ulterior para continuar con la consulta. Además, si el tamaño del conjunto de resultados supera 1 MB antes de que DynamoDB alcance este límite, detiene la consulta y	No

Nombre	Descripción	Obligatorio
	devuelve los valores coinciden tes, junto con el `LastEvalu atedKey` , que puede aplicarse en una operación ulterior para continuar con la consulta. Tipo: Number	
`ConsistentRead`	Si se establece en `true`, se emite una lectura consisten te; en caso contrario, se utiliza una lectura consistente final. Tipo: Booleano	No

Nombre	Descripción	Obligatorio
`Count`	Si se establece en `true`, DynamoDB devuelve la cantidad total de elementos que coinciden con los parámetros de la consulta, en lugar de una lista de elementos coincidentes y sus atributos. Puede aplicar el parámetro `Limit` a las consultas que son solo de recuento. No establezca `Count` en `true` si proporciona una lista de `AttributesToGet` ; si así lo hiciera, DynamoDB devolverá un error de validación. Para obtener más información, consulte Recuento de los elementos en los resultados. Tipo: Booleano	No
`HashKeyValue`	Valor de atributo del componente hash de la clave principal compuesta. Tipo: String, Number o Binary	Sí

Nombre	Descripción	Obligatorio
`RangeKeyCondition`	Contenedor de los valores de los atributos y operadores de comparación que se usarán en la consulta. Una solicitud de consulta no requiere una condición `RangeKeyCondition`. Si se proporciona solo el `HashKeyValue`, DynamoDB devuelve todos los elementos que tienen el valor especificado de clave hash. Tipo: mapa	No
`RangeKeyCondition : AttributeValueList`	Valores de los atributos que se van a evaluar para los parámetros de la consulta. La `AttributeValueList` contiene un solo valor de atributo, a no ser que se especifique una comparación `BETWEEN`. Para la comparación `BETWEEN`, la lista `AttributeValueList` contiene dos valores de atributos. Tipo: Map, mapa de `AttributeValue` a un `ComparisonOperator`.	No

Nombre	Descripción	Obligatorio
`RangeKeyCondition : ComparisonOperator`	Criterios para evaluar los atributos proporcionados, tales como igual que, mayor que, etc. A continuación se muestran los operadores de comparación válidos para una operación Query. **Note** Las comparaciones de valores de cadenas de tipo mayor que, igual que o menor que se basan en sus valores según el código de caracteres ASCII. Por ejemplo, `a` es mayor que `A` y `aa` es mayor que `B`. Para obtener una lista de valores de códigos, consulte http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters. En el tipo Binary, al comparar valores binarios DynamoDB trata cada byte como datos sin signo; por ejemplo, al evaluar expresiones de consulta.	No

Nombre	Descripción	Obligatorio
	Tipo: String o Binary	
	`EQ`: igual que. Para `EQ`, `AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si un elemento contiene un `AttributeValue` de un tipo distinto del especificado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}` . `{"N":"6"}` tampoco es igual que `{"NS":["6", "2", "1"]}` .	

Nombre	Descripción	Obligatorio
	`LE`: menor o igual que. Para `LE`, `AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si un elemento contiene un `AttributeValue` de un tipo distinto del especificado en la solicitud , el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}` . `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.	

Nombre	Descripción	Obligatorio
	`LT` : Menor que. Para `LT`, `Attribute ValueList` puede contener un solo `Attribute Value` de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si un elemento contiene un `Attribute Value` de un tipo distinto del especificado en la solicitud , el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}` . `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.	

Nombre	Descripción	Obligatorio
	`GE`: mayor o igual que. Para `GE`, `AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si un elemento contiene un `AttributeValue` de un tipo distinto del especificado en la solicitud , el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}` . `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.	

Nombre	Descripción	Obligatorio
	`GT` : Mayor que. Para `GT`, `AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si un elemento contiene un `AttributeValue` de un tipo distinto del especificado en la solicitud , el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}` . `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.	
	`BEGINS_WITH` : comprueba si hay un prefijo. Para `BEGINS_WITH` , `AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena) o Binary (binario) (no de tipo Number [número] ni un conjunto). El atributo de destino de la comparación debe ser un valor de tipo String o Binary (no de tipo Number ni un conjunto).	

Nombre	Descripción	Obligatorio
	`BETWEEN`: mayor o igual que el primer valor y menor o igual que el segundo valor. Para `BETWEEN`, `AttributeValueList` debe contener dos elemento `AttributeValue` del mismo tipo, que puede ser String (cadena), Number (número) o Binary (binario) (pero no un conjunto). Un atributo de destino coincide si el valor de destino es mayor o igual que el primer elemento y menor o igual que el segundo elemento. Si un elemento contiene un `AttributeValue` de un tipo distinto del especificado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no se compara con `{"N":"6"}`. `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.	

Nombre	Descripción	Obligatorio
`ScanIndexForward`	Especifica si el índice se recorrerá en sentido ascendente o descenden te. DynamoDB devuelve los resultados según el orden solicitado en función de la clave de rango: si el tipo de datos es Number (número), los resultados se devuelven en orden numérico; de lo contrario, el recorrido se basa en los valores del código de caracteres ASCII. Tipo: Booleano El valor predeterminado es `true` (ascendente).	No

Nombre	Descripción	Obligatorio
`ExclusiveStartKey`	Clave principal del elemento a partir del cual se continuará una consulta anterior. Una consulta anterior puede proporcionar este valor en `LastEvaluatedKey` si esa operación se interrumpió antes de completar la consulta, ya sea debido al tamaño del conjunto de resultados o al parámetro `Limit`. El valor de `LastEvaluatedKey` se puede pasar a una nueva solicitud de consulta para continuar la operación a partir de ese punto. Tipo: `HashKeyElement`, o bien `HashKeyElement` y `RangeKeyElement` si se trata de una clave principal compuesta.	No

Respuestas

Sintaxis

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 308

{"Count":2,"Items":[{
    "AttributeName1":{"S":"AttributeValue1"},
    "AttributeName2":{"N":"AttributeValue2"},
```

```
    "AttributeName3":{"S":"AttributeValue3"}
    },{
    "AttributeName1":{"S":"AttributeValue3"},
    "AttributeName2":{"N":"AttributeValue4"},
    "AttributeName3":{"S":"AttributeValue3"},
    "AttributeName5":{"B":"dmFsdWU="}
}],
    "LastEvaluatedKey":{"HashKeyElement":{"AttributeValue3":"S"},
                        "RangeKeyElement":{"AttributeValue4":"N"}
     },
     "ConsumedCapacityUnits":1
}
```

Nombre	Descripción
`Items`	Atributos de elementos que cumplen los parámetros de la consulta. Tipo: Map, mapa de los nombres de atributos y sus tipos de datos y valores.
`Count`	Cantidad de elementos de la respuesta. Para obtener más información, consulte Recuento de los elementos en los resultados. Tipo: Number
`LastEvaluatedKey`	Clave principal del elemento en el que se ha detenido la operación de consulta, incluido el conjunto de resultados anterior. Utilice este valor para iniciar una nueva operación excluyendo este valor en la nueva solicitud. El valor de `LastEvaluatedKey` es `null` cuando se ha completado todo el conjunto de resultados de la consulta (es decir, cuando la operación ha procesado la “última página”).

Nombre	Descripción
	Tipo: `HashKeyElement` , o bien `HashKeyElement` y `RangeKeyElement` si se trata de una clave principal compuesta.
`ConsumedCapacityUnits`	Cantidad de unidades de capacidad de lectura consumidas por la operación. Este valor muestra el número aplicado al rendimiento aprovisionado. Para obtener más información, consulte Modo de capacidad aprovisionada de DynamoDB. Tipo: Number

Errores especiales

Error	Descripción
`ResourceNotFoundException`	No se encontró la tabla especificada.

Ejemplos

Para obtener ejemplos sobre cómo usar el SDK de AWS, consulte Consulta de tablas en DynamoDB.

Solicitud de ejemplo

```
// This header is abbreviated. For a sample of a complete header, see API de bajo nivel
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable",
 "Limit":2,
 "HashKeyValue":{"S":"John"},
 "ScanIndexForward":false,
 "ExclusiveStartKey":{
```

```
  "HashKeyElement":{"S":"John"},
  "RangeKeyElement":{"S":"The Matrix"}
 }
}
```

Respuesta de ejemplo

```
HTTP/1.1 200
x-amzn-RequestId: 3647e778-71eb-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 308

{"Count":2,"Items":[{
 "fans":{"SS":["Jody","Jake"]},
 "name":{"S":"John"},
 "rating":{"S":"***"},
 "title":{"S":"The End"}
 },{
 "fans":{"SS":["Jody","Jake"]},
 "name":{"S":"John"},
 "rating":{"S":"***"},
 "title":{"S":"The Beatles"}
 }],
 "LastEvaluatedKey":{"HashKeyElement":{"S":"John"},"RangeKeyElement":{"S":"The
 Beatles"}},
"ConsumedCapacityUnits":1
}
```

Solicitud de ejemplo

```
// This header is abbreviated. For a sample of a complete header, see API de bajo nivel
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Query
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable",
 "Limit":2,
 "HashKeyValue":{"S":"Airplane"},
 "RangeKeyCondition":{"AttributeValueList":[{"N":"1980"}],"ComparisonOperator":"EQ"},
 "ScanIndexForward":false}
```

Respuesta de ejemplo

```
HTTP/1.1 200
x-amzn-RequestId: 8b9ee1ad-774c-11e0-9172-d954e38f553a
content-type: application/x-amz-json-1.0
content-length: 119

{"Count":1,"Items":[{
 "fans":{"SS":["Dave","Aaron"]},
 "name":{"S":"Airplane"},
 "rating":{"S":"***"},
 "year":{"N":"1980"}
 }],
"ConsumedCapacityUnits":1
}
```

Acciones relacionadas

- Examen

Examen

> **Important**
>
> ***Esta sección se refiere a la versión 2011-12-05 del API, que está obsoleta y no debe utilizarse para nuevas aplicaciones.***
>
> Para consultar la documentación sobre la API de bajo nivel actual, consulte la Referencia de la API de Amazon DynamoDB.

Descripción

La operación `Scan` lleva a cabo un examen completo de una tabla y devuelve uno o varios elementos y sus atributos. Proporcione un filtro `ScanFilter` para obtener resultados más específicos.

> **Note**
>
> Si la cantidad total de elementos examinados supera el límite de 1 MB, el examen se detiene y se devuelven los resultados al usuario; en este caso, se facilita un `LastEvaluatedKey`

para que pueda continuar el examen en una operación posterior. Además, los resultados incluyen la cantidad de elementos que superan el límite. En un examen, puede suceder que ninguno de los datos de la tabla cumplan los criterios de filtro.
El conjunto de resultados presenta consistencia final.

Solicitudes

Sintaxis

```
// This header is abbreviated.
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Limit": 2,
    "ScanFilter":{
        "AttributeName1":{"AttributeValueList":
[{"S":"AttributeValue"}],"ComparisonOperator":"EQ"}
    },
    "ExclusiveStartKey":{
        "HashKeyElement":{"S":"AttributeName1"},
        "RangeKeyElement":{"N":"AttributeName2"}
    },
    "AttributesToGet":["AttributeName1", "AttributeName2", "AttributeName3"]},
}
```

Nombre	Descripción	Obligatorio
`TableName`	Nombre de la tabla que contiene los elementos solicitados. Tipo: cadena	Sí
`AttributesToGet`	Matriz de nombres de atributo. Si no se especifican sus nombres, se devuelven todos	No

Nombre	Descripción	Obligatorio
	los atributos. Si algún atributo no se encuentra, no aparecerá en los resultados. Tipo: matriz	
`Limit`	Número máximo de elementos que se van a evaluar, que no es necesariamente el número de elementos coincidentes. Si DynamoDB alcanza el límite de cantidad de elementos mientras procesa los resultados, se detiene y devuelve los valores coincidentes hasta ese punto, junto con un `LastEvaluatedKey` que puede aplicarse en una operación ulterior para continuar recuperando elementos. Además, si el tamaño del conjunto de datos examinados supera 1 MB antes de que DynamoDB alcance este límite, detiene el examen y devuelve los valores coincidentes hasta el límite, junto con `LastEvaluatedKey` que puede aplicarse en una operación ulterior para continuar con el examen. Tipo: Number	No

Nombre	Descripción	Obligatorio
`Count`	Si se establece en `true`, DynamoDB devuelve la cantidad total de elementos de la operación Scan, aunque no encuentre ningún elemento coincidente para el filtro asignado. Puede aplicar el parámetro Limit a los exámenes que son solo de recuento. No establezca `Count` en `true` si proporciona una lista de `AttributesToGet` ; si así lo hiciera, DynamoDB devolverá un error de validación. Para obtener más información, consulte Recuento de los elementos en los resultados. Tipo: Booleano	No
`ScanFilter`	Evalúa los resultados del examen y solo devuelve los valores deseados. Si se especifican varias condicion es, se tratan como operacion es "AND": deberán cumplirse todas las condiciones para que se incluyan en los resultados. Tipo: Un mapa de nombres de atributos a valores con operadores de comparación.	No

Nombre	Descripción	Obligatorio
`ScanFilter` :`AttributeValueList`	Valores y condiciones que se usarán para evaluar los resultados del examen para el filtro. Tipo: Map, mapa de `AttributeValue` a un `Condition` .	No

Nombre	Descripción	Obligatorio
`ScanFilter : ComparisonOperator`	Criterios para evaluar los atributos proporcionados, tales como igual que, mayor que, etc. A continuación se muestran los operadores de comparación válidos para una operación de examen. **Note** Las comparaciones de valores de cadenas de tipo mayor que, igual que o menor que se basan en sus valores según el código de caracteres ASCII. Por ejemplo, `a` es mayor que `A` y `aa` es mayor que `B`. Para obtener una lista de valores de códigos, consulte http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters. En el tipo Binary, al comparar valores binarios DynamoDB trata cada byte como datos sin signo; por ejemplo, al evaluar expresiones de consulta.	No

Nombre	Descripción	Obligatorio
	Tipo: String o Binary	
	`EQ`: igual que. Para `EQ`, `AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si un elemento contiene un `AttributeValue` de un tipo distinto del especificado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}` . `{"N":"6"}` tampoco es igual que `{"NS":["6", "2", "1"]}` .	

Nombre	Descripción	Obligatorio
	NE: distinto de. Para NE, `Attribute ValueList` puede contener un solo `Attribute Value` de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si un elemento contiene un `Attribute Value` de un tipo distinto del especificado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}` . `{"N":"6"}` tampoco es igual que `{"NS": ["6", "2", "1"]}` .	

Nombre	Descripción	Obligatorio
	`LE`: menor o igual que. Para `LE`, `Attribute ValueList` puede contener un solo `Attribute Value` de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si un elemento contiene un `Attribute Value` de un tipo distinto del especificado en la solicitud , el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}` . `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.	

Nombre	Descripción	Obligatorio
	`LT` : Menor que. Para `LT`, `AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si un elemento contiene un `AttributeValue` de un tipo distinto del especificado en la solicitud , el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}` . `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.	

Nombre	Descripción	Obligatorio
	`GE`: mayor o igual que. Para `GE`, `Attribute ValueList` puede contener un solo `Attribute Value` de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si un elemento contiene un `Attribute Value` de un tipo distinto del especificado en la solicitud , el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}` . `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.	

Nombre	Descripción	Obligatorio
	`GT` : Mayor que. Para `GT`, `AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si un elemento contiene un `AttributeValue` de un tipo distinto del especificado en la solicitud , el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}` . `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.	
	`NOT_NULL`: el atributo existe.	
	`NULL`: el atributo no existe.	

Nombre	Descripción	Obligatorio
	`CONTAINS`: comprueba si hay una subsecuencia, o un valor en un conjunto. Para `CONTAINS`, `AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si el atributo de destino de la comparación es de tipo String (cadena), entonces la operación comprueba si hay una subcadena coincidente. Si el atributo de destino de la comparación es de tipo Binary (binario), entonces la operación busca una subsecuencia del destino que coincida con la entrada. Si el atributo de destino de la comparación es un conjunto ("SS", "NS" o "BS"), entonces la operación busca un miembro del conjunto (no como subcadena).	

Nombre	Descripción	Obligatorio
	`NOT_CONTAINS` : comprueba la ausencia de una subsecuencia, o la ausencia de un valor en un conjunto. Para `NOT_CONTAINS` , `AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si el atributo de destino de la comparación es de tipo String (cadena), entonces la operación verifica la ausencia de una subcadena coincidente. Si el atributo de destino de la comparación es de tipo Binary (binario), entonces la operación verifica la ausencia de una subsecuencia del destino que coincida con la entrada. Si el atributo de destino de la comparación es un conjunto ("SS", "NS" o "BS"), entonces la operación busca la ausencia de un miembro del conjunto (no como subcadena).	

Nombre	Descripción	Obligatorio
	`BEGINS_WITH` : comprueba si hay un prefijo. Para `BEGINS_WITH` , `AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena) o Binary (binario) (no de tipo Number [número] ni un conjunto). El atributo de destino de la comparación debe ser un valor de tipo String o Binary (no de tipo Number ni un conjunto).	
	`IN`: comprueba si hay coincidencias exactas. Para `IN`, `AttributeValueList` puede contener más de un `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). El atributo de destino de la comparación debe ser del mismo tipo y valor exactos para que se produzca la coincidencia. Un valor de tipo String nunca coincide con un valor de tipo String Set.	

Nombre	Descripción	Obligatorio
	`BETWEEN`: mayor o igual que el primer valor y menor o igual que el segundo valor. Para `BETWEEN`, `AttributeValueList` debe contener dos elemento `AttributeValue` del mismo tipo, que puede ser String (cadena), Number (número) o Binary (binario) (pero no un conjunto). Un atributo de destino coincide si el valor de destino es mayor o igual que el primer elemento y menor o igual que el segundo elemento. Si un elemento contiene un `AttributeValue` de un tipo distinto del especificado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no se compara con `{"N":"6"}`. `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.	

Nombre	Descripción	Obligatorio
`ExclusiveStartKey`	Clave principal del elemento a partir del cual se continuará un examen anterior. Un examen anterior puede proporcionar este valor si la operación de examen se interrumpió antes de examinar toda la tabla, ya sea debido al tamaño del conjunto de resultados o al parámetro `Limit`. El valor de `LastEvaluatedKey` se puede pasar a una nueva solicitud de examen para continuar la operación a partir de ese punto. Tipo: `HashKeyElement` , o bien `HashKeyElement` y `RangeKeyElement` si se trata de una clave principal compuesta.	No

Respuestas

Sintaxis

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 229

{"Count":2,"Items":[{
    "AttributeName1":{"S":"AttributeValue1"},
    "AttributeName2":{"S":"AttributeValue2"},
    "AttributeName3":{"S":"AttributeValue3"}
    },{
```

```
    "AttributeName1":{"S":"AttributeValue4"},
    "AttributeName2":{"S":"AttributeValue5"},
    "AttributeName3":{"S":"AttributeValue6"},
    "AttributeName5":{"B":"dmFsdWU="}
    }],
    "LastEvaluatedKey":
        {"HashKeyElement":{"S":"AttributeName1"},
        "RangeKeyElement":{"N":"AttributeName2"},
    "ConsumedCapacityUnits":1,
    "ScannedCount":2}
}
```

Nombre	Descripción
`Items`	Contenedor de los atributos que coinciden con los parámetros de la operación. Tipo: Map, mapa de los nombres de atributos y sus tipos de datos y valores.
`Count`	Cantidad de elementos de la respuesta. Para obtener más información, consulte Recuento de los elementos en los resultados. Tipo: Number
`ScannedCount`	Cantidad de elementos del examen completo antes de aplicar cualquier filtro. Un valor de `ScannedCount` elevado, con pocos resultados de `Count` o ninguno indica que la operación Scan ha sido ineficiente. Para obtener más información, consulte Recuento de los elementos en los resultados. Tipo: Number
`LastEvaluatedKey`	Clave principal del elemento en la que se detuvo la operación de examen. Proporcio ne este valor en una operación de examen

Nombre	Descripción
	posterior para continuar la operación a partir de ese punto. El valor de `LastEvaluatedKey` es `null` cuando se ha completado todo el conjunto de resultados de examen (es decir, cuando la operación ha procesado la "última página").
`ConsumedCapacityUnits`	Cantidad de unidades de capacidad de lectura consumidas por la operación. Este valor muestra el número aplicado al rendimiento aprovisionado. Para obtener más información, consulte Modo de capacidad aprovisionada de DynamoDB. Tipo: Number

Errores especiales

Error	Descripción
`ResourceNotFoundException`	No se encontró la tabla especificada.

Ejemplos

Para obtener ejemplos sobre cómo usar el SDK de AWS, consulte Análisis de tablas en DynamoDB.

Solicitud de ejemplo

```
// This header is abbreviated. For a sample of a complete header, see API de bajo nivel
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"1-hash-rangetable","ScanFilter":{}}
```

Respuesta de ejemplo

```
HTTP/1.1 200
x-amzn-RequestId: 4e8a5fa9-71e7-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 465

{"Count":4,"Items":[{
 "date":{"S":"1980"},
 "fans":{"SS":["Dave","Aaron"]},
 "name":{"S":"Airplane"},
 "rating":{"S":"***"}
 },{
 "date":{"S":"1999"},
 "fans":{"SS":["Ziggy","Laura","Dean"]},
 "name":{"S":"Matrix"},
 "rating":{"S":"*****"}
 },{
 "date":{"S":"1976"},
 "fans":{"SS":["Riley"]},"
 name":{"S":"The Shaggy D.A."},
 "rating":{"S":"**"}
 },{
 "date":{"S":"1985"},
 "fans":{"SS":["Fox","Lloyd"]},
 "name":{"S":"Back To The Future"},
 "rating":{"S":"****"}
 }],
    "ConsumedCapacityUnits":0.5
 "ScannedCount":4}
```

Solicitud de ejemplo

```
// This header is abbreviated. For a sample of a complete header, see API de bajo nivel
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0
content-length: 125

{"TableName":"comp5",
 "ScanFilter":
  {"time":
```

```
   {"AttributeValueList":[{"N":"400"}],
   "ComparisonOperator":"GT"}
 }
}
```

Respuesta de ejemplo

```
HTTP/1.1 200 OK
x-amzn-RequestId: PD1CQK9QCTERLTJP20VALJ60TRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 262
Date: Mon, 15 Aug 2011 16:52:02 GMT

{"Count":2,
 "Items":[
  {"friends":{"SS":["Dave","Ziggy","Barrie"]},
  "status":{"S":"chatting"},
  "time":{"N":"2000"},
  "user":{"S":"Casey"}},
  {"friends":{"SS":["Dave","Ziggy","Barrie"]},
  "status":{"S":"chatting"},
  "time":{"N":"2000"},
  "user":{"S":"Fredy"}
  }],
"ConsumedCapacityUnits":0.5
"ScannedCount":4
}
```

Solicitud de ejemplo

```
// This header is abbreviated. For a sample of a complete header, see API de bajo nivel
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.Scan
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
 "Limit":2,
 "ScanFilter":
  {"time":
   {"AttributeValueList":[{"N":"400"}],
   "ComparisonOperator":"GT"}
 },
```

```
 "ExclusiveStartKey":
  {"HashKeyElement":{"S":"Fredy"},"RangeKeyElement":{"N":"2000"}}
}
```

Respuesta de ejemplo

```
HTTP/1.1 200 OK
x-amzn-RequestId: PD1CQK9QCTERLTJP20VALJ60TRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 232
Date: Mon, 15 Aug 2011 16:52:02 GMT

{"Count":1,
 "Items":[
  {"friends":{"SS":["Jane","James","John"]},
  "status":{"S":"exercising"},
  "time":{"N":"2200"},
  "user":{"S":"Roger"}}
 ],
 "LastEvaluatedKey":{"HashKeyElement":{"S":"Riley"},"RangeKeyElement":{"N":"250"}},
"ConsumedCapacityUnits":0.5
"ScannedCount":2
}
```

Acciones relacionadas

- Consultar
- BatchGetItem

UpdateItem

> **Important**
>
> ***Esta sección se refiere a la versión 2011-12-05 del API, que está obsoleta y no debe utilizarse para nuevas aplicaciones.***
>
> Para consultar la documentación sobre la API de bajo nivel actual, consulte la Referencia de la API de Amazon DynamoDB.

Descripción

Edita los atributos de un elemento existente. Puede realizar una actualización condicional, es decir, insertar un nuevo par nombre-valor de atributo si no existe, o bien sustituir un par de nombre-valor si tiene determinados valores de atributo esperados.

> **Note**
>
> Los atributos de clave principal no se pueden actualizar mediante UpdateItem. En lugar de ello, debe eliminar el elemento y utilizar PutItem para crear otro con los nuevos atributos.

La operación UpdateItem incluye un parámetro `Action`, que define cómo realizar la actualización. Puede poner, eliminar o agregar valores de atributos.

Los valores de los atributos no pueden ser null; los atributos de tipo cadena y binario deben tener una longitud superior a cero; y los atributos de tipo conjunto no pueden estar vacíos. Las solicitudes con valores vacíos se rechazan con la excepción `ValidationException`.

Si un elemento tiene la clave principal especificada:

- PUT: agrega el atributo especificado. Si el atributo existe, se sustituye por el nuevo valor.
- DELETE: si no se especifica ningún valor, esta acción elimina el atributo y su valor. Si se especifica un conjunto de valores, entonces los valores del conjunto especificado se eliminan del conjunto anterior. Por lo tanto, si el valor del atributo contiene [a,b,c] y la acción de eliminación contiene [a,c], el valor del atributo final es [b]. El tipo del valor especificado debe coincidir con el tipo de valor existente. No es válido especificar un conjunto vacío.
- ADD: la acción de añadir se utiliza sólo para números o si el atributo de destino es un conjunto (incluidos los conjuntos de cadena). ADD no funciona si el atributo de destino es un valor de cadena única o un valor escalar de tipo Binary (binario). El valor especificado se suma a un valor numérico (se suma o se resta al valor numérico existente) o se agrega como valor adicional a un conjunto de cadenas. Si se especifica un conjunto de valores, los valores se agregan al conjunto existente. Por ejemplo, si el conjunto original es [1,2] y el valor suministrado es [3], después de la operación de añadir el conjunto será [1,2,3] y no [4,5]. Se produce un error si se especifica una acción Add para un atributo de tipo Set y el tipo de atributo especificado no coincide con el tipo de conjunto existente.

  Si utiliza ADD para un atributo que no existe, el atributo y sus valores se agregan al elemento.

Si no hay ningún elemento que coincida con la clave principal especificada:

- PUT: crea un elemento nuevo con la clave principal especificada. A continuación, agrega el atributo especificado.
- DELETE: no sucede nada.
- ADD: crea un elemento con la clave principal y el número (o el conjunto de números) suministrados para el valor del atributo. No es válido para los tipos de cadena o binario.

> **Note**
>
> Si utiliza ADD para sumar o restar de un valor numérico de un elemento que no existía antes de la actualización, DynamoDB utilizará `0` como valor inicial. Además, si actualiza un elemento mediante ADD para sumar o restar de un valor numérico de un atributo que no existía antes de la actualización (pero el elemento sí existía), DynamoDB utilizará `0` como valor inicial. Por ejemplo, si utiliza ADD para sumar `+3` a un atributo que no existía antes de la actualización. DynamoDB utilizará `0` para el valor inicial y el valor posterior a la actualización será `3`.

Para obtener más información sobre cómo usar esta operación, consulte Uso de elementos y atributos en DynamoDB.

Solicitudes

Sintaxis

```
// This header is abbreviated.
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateItem
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "Key":
        {"HashKeyElement":{"S":"AttributeValue1"},
        "RangeKeyElement":{"N":"AttributeValue2"}},
    "AttributeUpdates":{"AttributeName3":{"Value":
{"S":"AttributeValue3_New"},"Action":"PUT"}},
    "Expected":{"AttributeName3":{"Value":{"S":"AttributeValue3_Current"}}},
```

```
    "ReturnValues":"ReturnValuesConstant"
}
```

Nombre	Descripción	Obligatorio
`TableName`	Nombre de la tabla que contiene el elemento que se va a actualizar. Tipo: cadena	Sí
`Key`	Clave principal que define el elemento. Para obtener más información sobre claves principales, consulte Clave principal. Tipo: Map, mapa de `HashKeyElement` a su valor y `RangeKeyElement` a su valor.	Sí
`AttributeUpdates`	Mapa del nombre de atributo al nuevo valor y acción de la actualización. Los nombres de los atributos especific an los atributos que hay que modificar y no pueden contener atributos de clave principal. Tipo: Map, mapa de nombre de atributo, valor y acción de la actualización del atributo.	
`Attribute Updates :Action`	Especifica cómo realizar la actualización. Valores posibles: PUT (predeter	No

Nombre	Descripción	Obligatorio
	minado), `ADD` o `DELETE`. La semántica se explica en la descripción de UpdateItem. Tipo: cadena Valor predeterminado: `PUT`	
`Expected`	Designa un atributo para una actualización condicional. El parámetro `Expected` le permite proporcionar un nombre de atributo e indicar si DynamoDB debe verificar si el valor del atributo ya existe; o bien si el valor del atributo existe y tiene un valor determinado antes de cambiarlo. Tipo: Map, mapa de nombres de atributos.	No
`Expected:AttributeName`	Nombre del atributo para la operación Put condicional. Tipo: cadena	No

Nombre	Descripción	Obligatorio

`Expected:AttributeName: ExpectedAttributeValue`	Use este parámetro para especificar si ya existe un valor del par de nombre-valor del atributo.
	
	En la siguiente notación JSON, se actualiza el elemento si todavía no existe el atributo "Color" para ese elemento:
	
	`"Expected" : {"Color":{"Exists":false}}`
	
	En la siguiente notación JSON se comprueba si el atributo denominado "Color" tiene el valor "Yellow" antes de actualizar el elemento:
	
	`"Expected" : {"Color":{"Exists":true},{"Value":{"S":"Yellow"}}}`
	
	De forma predeterminada, si utiliza el parámetro `Expected` y le proporciona un `Value`, DynamoDB da por hecho que el atributo existe y que posee un valor que hay que sustituir. Por lo tanto, no es preciso especificar `{"Exists":true}` , porque
		No

Nombre	Descripción	Obligatorio
	se considera implícito. Puede reducir la solicitud a: `"Expected" : {"Color":{"Value":{"S":"Yellow"}}}` **Note** Si especifica `{"Exists":true}` sin un valor de atributo que verificar, DynamoDB devuelve un error.	

Nombre	Descripción	Obligatorio
`ReturnValues`	Use este parámetro si desea obtener los pares de nombre-valor de los atributos antes de actualizarlos mediante la solicitud `UpdateItem`. Los valores posibles de los parámetros son `NONE` (predeterminado) o `ALL_OLD`, `UPDATED_OLD`, `ALL_NEW` o `UPDATED_NEW`. Si se especifica `ALL_OLD` y `UpdateItem` ha sobrescrito un par de nombre-valor del atributo, se devuelve el contenido del elemento anterior. Si este parámetro no se proporciona o si su valor es `NONE`, no se devuelve nada. Si se especifica `ALL_NEW`, entonces se devuelven todos los atributos de la nueva versión del elemento. Si se especifica `UPDATED_NEW`, entonces se devuelven solamente las nuevas versiones de los atributos actualizados. Tipo: cadena	No

Respuestas

Sintaxis

En el ejemplo de sintaxis siguiente se supone que la solicitud ha especificado un parámetro `ReturnValues` de `ALL_OLD`; en caso contrario, la respuesta solo contiene la entrada `ConsumedCapacityUnits`.

```
HTTP/1.1 200
x-amzn-RequestId: 8966d095-71e9-11e0-a498-71d736f27375
content-type: application/x-amz-json-1.0
content-length: 140

{"Attributes":{
 "AttributeName1":{"S":"AttributeValue1"},
 "AttributeName2":{"S":"AttributeValue2"},
 "AttributeName3":{"S":"AttributeValue3"},
 "AttributeName5":{"B":"dmFsdWU="}
 },
"ConsumedCapacityUnits":1
}
```

Nombre	Descripción
`Attributes`	Mapa de pares de nombre-valor del atributo, pero solo si el parámetro `ReturnValues` se ha especificado como algo distinto de `NONE` en la solicitud. Tipo: Map, mapa de pares de nombre-valor del atributo.
`ConsumedCapacityUnits`	Cantidad de unidades de capacidad de escritura consumidas por la operación. Este valor muestra el número aplicado al rendimien to aprovisionado. Para obtener más informaci ón, consulte Modo de capacidad aprovisionada de DynamoDB. Tipo: Number

Errores especiales

Error	Descripción
`ConditionalCheckFailedException`	Error en la verificación condicional. El valor del atributo ("+ name +") es ("+ value +") pero se esperaba ("+ expValue +").
`ResourceNotFoundExceptions`	No se encontró el elemento o atributo especific ado.

Ejemplos

Para obtener ejemplos sobre cómo usar el SDK de AWS, consulte Uso de elementos y atributos en DynamoDB.

Solicitud de ejemplo

```
// This header is abbreviated. For a sample of a complete header, see API de bajo nivel
 de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateItem
content-type: application/x-amz-json-1.0

{"TableName":"comp5",
    "Key":
        {"HashKeyElement":{"S":"Julie"},"RangeKeyElement":{"N":"1307654350"}},
    "AttributeUpdates":
        {"status":{"Value":{"S":"online"},
        "Action":"PUT"}},
    "Expected":{"status":{"Value":{"S":"offline"}}},
    "ReturnValues":"ALL_NEW"
}
```

Respuesta de ejemplo

```
HTTP/1.1 200 OK
x-amzn-RequestId: 5IMHO7F01Q9P7Q6QMKMMI3R3QRVV4KQNSO5AEMVJF66Q9ASUAAJG
content-type: application/x-amz-json-1.0
content-length: 121
```

```
Date: Fri, 26 Aug 2011 21:05:00 GMT

{"Attributes":
    {"friends":{"SS":["Lynda, Aaron"]},
    "status":{"S":"online"},
    "time":{"N":"1307654350"},
    "user":{"S":"Julie"}},
"ConsumedCapacityUnits":1
}
```

Acciones relacionadas

- PutItem
- DeleteItem

UpdateTable

> **Important**
>
> ***Esta sección se refiere a la versión 2011-12-05 del API, que está obsoleta y no debe utilizarse para nuevas aplicaciones.***
>
> Para consultar la documentación sobre la API de bajo nivel actual, consulte la Referencia de la API de Amazon DynamoDB.

Descripción

Actualiza el desempeño aprovisionado de la tabla en cuestión. Configurar el rendimiento de una tabla para le ayuda a su administración; esto forma parte de la característica de rendimiento aprovisionado de DynamoDB. Para obtener más información, consulte Modo de capacidad aprovisionada de DynamoDB.

Los valores de desempeño provisionado se pueden aumentar o reducir según los máximos o mínimos indicados en Cuotas en Amazon DynamoDB.

La tabla debe encontrarse en el estado `ACTIVE` para poder realizar correctamente esta operación. UpdateTable es una operación asíncrona; es decir, mientras se ejecuta la operación, la tabla se mantiene en el estado `UPDATING`. Mientras la tabla se encuentra en el estado `UPDATING`, sigue teniendo el rendimiento provisionado previo a la llamada. La nueva configuración de desempeño

provisionado no entra en vigor hasta que la tabla vuelve al estado `ACTIVE` después de la operación UpdateTable.

Solicitudes

Sintaxis

```
// This header is abbreviated.
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateTable
content-type: application/x-amz-json-1.0

{"TableName":"Table1",
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":15}
}
```

Nombre	Descripción	Obligatorio
`TableName`	Nombre de la tabla que se va a actualizar. Tipo: cadena	Sí
`ProvisionedThroughput`	Nuevo rendimiento de la tabla especificada, que se compone de los valores de `ReadCapacityUnits` y `WriteCapacityUnits`. Consulte Modo de capacidad aprovisionada de DynamoDB. Tipo: matriz	Sí
`ProvisionedThroughput:ReadCapacityUnits`	Establece el número mínimo de `ReadCapacityUnits` consistentes consumidas por segundo para la tabla especificada antes de que	Sí

Nombre	Descripción	Obligatorio
	DynamoDB equilibre la carga con otras operaciones. Las operaciones de lectura consistente final requieren menos esfuerzo que una operación de lectura consisten te; por lo tanto, un ajuste de 50 unidades `ReadCapac ityUnits` consistentes por segundo proporciona 100 unidades de capacidad `ReadCapacityUnits` consistentes finales por segundo. Tipo: Number	
`ProvisionedThrough put :WriteCapa cityUnits`	Establece el número mínimo de `WriteCapacityUnits` consumidas por segundo para la tabla especificada antes de que DynamoDB equilibre la carga con otras operaciones. Tipo: Number	Sí

Respuestas

Sintaxis

```
HTTP/1.1 200 OK
x-amzn-RequestId: CSOC7TJPLR0OOKIRLGOHVAICUFVV4KQNSO5AEMVJF66Q9ASUAAJG
Content-Type: application/json
Content-Length: 311
Date: Tue, 12 Jul 2011 21:31:03 GMT
```

```
{"TableDescription":
    {"CreationDateTime":1.321657838135E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"AttributeValue1","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"AttributeValue2","AttributeType":"N"}},
    "ProvisionedThroughput":
        {"LastDecreaseDateTime":1.321661704489E9,
        "LastIncreaseDateTime":1.321663607695E9,
        "ReadCapacityUnits":5,
        "WriteCapacityUnits":10},
    "TableName":"Table1",
    "TableStatus":"UPDATING"}}
```

Nombre	Descripción
`CreationDateTime`	Fecha de creación de la tabla. Tipo: Number
`KeySchema`	Estructura de la clave principal (simple o compuesta) de la tabla. Se requiere un par de nombre-valor de `HashKeyElement` , pero el par de nombre-valor de `RangeKeyElement` es opcional (solo es obligatorio para las claves principales compuestas). El tamaño máximo de la clave hash es de 2048 bytes. El tamaño máximo de la clave de rango es de 1024 bytes. Ambos límites se aplican por separado (es decir, puede disponer de una clave combinada de rango y hash de 2048+1024). Para obtener más información sobre claves principales, consulte Clave principal. Tipo: Map, mapa de `HashKeyElement` , o bien de `HashKeyElement` y de `RangeKeyElement` si se trata de una clave principal compuesta.

Nombre	Descripción
`ProvisionedThroughput`	Ajustes de desempeño actuales de la tabla especificada, incluidos los valores de `LastIncreaseDateTime` (si procede) y `LastDecreaseDateTime` (si procede). Tipo: matriz
`TableName`	Nombre de la tabla actualizada. Tipo: cadena
`TableStatus`	Estado actual de la tabla (`CREATING`, `ACTIVE`, `DELETING` o `UPDATING`), que debe ser `UPDATING`. Use la operación DescribeTables para comprobar el estado de la tabla. Tipo: cadena

Errores especiales

Error	Descripción
`ResourceNotFoundException`	No se encontró la tabla especificada.
`ResourceInUseException`	La tabla no se encuentra en el estado `ACTIVE`.

Ejemplos

Solicitud de ejemplo

```
// This header is abbreviated.
// For a sample of a complete header, see API de bajo nivel de DynamoDB.
POST / HTTP/1.1
x-amz-target: DynamoDB_20111205.UpdateTable
```

```
content-type: application/x-amz-json-1.0

{"TableName":"comp1",
    "ProvisionedThroughput":{"ReadCapacityUnits":5,"WriteCapacityUnits":15}
}
```

Respuesta de ejemplo

```
HTTP/1.1 200 OK
content-type: application/x-amz-json-1.0
content-length: 390
Date: Sat, 19 Nov 2011 00:46:47 GMT

{"TableDescription":
    {"CreationDateTime":1.321657838135E9,
    "KeySchema":
        {"HashKeyElement":{"AttributeName":"user","AttributeType":"S"},
        "RangeKeyElement":{"AttributeName":"time","AttributeType":"N"}},
    "ProvisionedThroughput":
        {"LastDecreaseDateTime":1.321661704489E9,
        "LastIncreaseDateTime":1.321663607695E9,
        "ReadCapacityUnits":5,
        "WriteCapacityUnits":10},
    "TableName":"comp1",
    "TableStatus":"UPDATING"}
}
```

Acciones relacionadas

- CreateTable
- DescribeTables
- DeleteTable

Parámetros condicionales de DynamoDB heredados

En este documento se proporciona información general de los parámetros condicionales heredados de DynamoDB y se recomienda utilizar los nuevos parámetros de expresión en su lugar. Incluye detalles sobre parámetros como AttributeToGet, AttributeUpdates, ConditionalOperator, Expected, KeyConditions, QueryFilter y ScanFilter, y proporciona ejemplos de cómo utilizar los nuevos parámetros de expresión como sustitutos.

> **Important**
>
> Le recomendamos que utilice los nuevos parámetros de expresión en lugar de estos parámetros heredados siempre que sea posible. Para obtener más información, consulte Uso de expresiones en DynamoDB.
>
> Además, DynamoDB no permite mezclar parámetros condicionales heredados con parámetros de expresión en una misma llamada. Por ejemplo, llamar a la operación `Query` con `AttributesToGet` y `ConditionExpression` dará lugar a un error.

En la siguiente tabla se muestran las operaciones de la API de DynamoDB que todavía admiten estos parámetros heredados, así como los parámetros de expresión que deben utilizarse en su lugar. Esta tabla puede resultarle útil si va a actualizar las aplicaciones de tal forma que utilicen parámetros de expresión a partir de ahora.

Si utiliza esta operación de la API...	Con estos parámetros heredados...	Use estos parámetros de expresión en su lugar
`BatchGetItem`	`AttributesToGet`	`ProjectionExpression`
`DeleteItem`	`Expected`	`ConditionExpression`
`GetItem`	`AttributesToGet`	`ProjectionExpression`
`PutItem`	`Expected`	`ConditionExpression`
`Query`	`AttributesToGet`	`ProjectionExpression`
	`KeyConditions`	`KeyConditionExpression`
	`QueryFilter`	`FilterExpression`
`Scan`	`AttributesToGet`	`ProjectionExpression`
	`ScanFilter`	`FilterExpression`
`UpdateItem`	`AttributeUpdates`	`UpdateExpression`

Si utiliza esta operación de la API...	Con estos parámetros heredados...	Use estos parámetros de expresión en su lugar
	`Expected`	`ConditionExpression`

En las secciones siguientes se proporciona más información acerca de los parámetros condicionales heredados.

Temas

- AttributesToGet (heredado)
- AttributeUpdates (heredado)
- ConditionalOperator (heredado)
- Expected (heredado)
- KeyConditions (heredado)
- QueryFilter (heredado)
- ScanFilter (heredado)
- Escritura de condiciones con parámetros heredados

AttributesToGet (heredado)

> **Note**
>
> Le recomendamos que utilice los nuevos parámetros de expresión en lugar de estos parámetros heredados siempre que sea posible. Para obtener más información, consulte Uso de expresiones en DynamoDB. Para obtener información específica sobre el nuevo parámetro que reemplaza a este, Use ProjectionExpression en su lugar.

El parámetro condicional heredado `AttributesToGet` es una matriz de uno o varios atributos que se recuperan de DynamoDB. Si no se proporcionan sus nombres, se devuelven todos los atributos. Si cualquiera de los atributos solicitados no se encuentra, no aparecerá en el resultado.

`AttributesToGet` permite recuperar atributos de tipo List o Map; sin embargo, no puede recuperar entradas individuales dentro de una lista o un mapa.

Tenga en cuenta que `AttributesToGet` no modifica el consumo de rendimiento aprovisionado. DynamoDB determina las unidades de capacidad de lectura consumidas según el tamaño de los elementos y no según la cantidad de datos que se devuelven a la aplicación.

Use ProjectionExpression en su lugar; ejemplo

Supongamos que desea recuperar un elemento de la tabla Music, pero no devolverlo completo, sino solo algunos de los atributos. Podría usar una solicitud `GetItem` con un parámetro `AttributesToGet` como en este ejemplo de la AWS CLI:

```
aws dynamodb get-item \
    --table-name Music \
    --attributes-to-get '["Artist", "Genre"]' \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }'
```

Puede utilizar `ProjectionExpression` en su lugar:

```
aws dynamodb get-item \
    --table-name Music \
    --projection-expression "Artist, Genre" \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }'
```

AttributeUpdates (heredado)

> **Note**
>
> Le recomendamos que utilice los nuevos parámetros de expresión en lugar de estos parámetros heredados siempre que sea posible. Para obtener más información, consulte Uso de expresiones en DynamoDB. Para obtener información específica sobre el nuevo parámetro que reemplaza a este, Use UpdateExpression en su lugar..

En una operación `UpdateItem`, el parámetro condicional heredado `AttributeUpdates` contiene los nombres de los atributos que se van a modificar, su nombre, la acción que se va a llevar a

cabo y su nuevo valor. Si va a actualizar un atributo de clave de índice de alguno de los índices de esa tabla, el tipo de atributo debe coincidir con el tipo de clave de índice definido en el elemento `AttributesDefinition` de la descripción de la tabla. Puede utilizar `UpdateItem` para actualizar los atributos sin clave.

Los valores de los atributos no pueden ser null. La longitud de los atributos de tipo String (cadena) y Binary (binario) debe ser mayor que cero. Los atributos de tipo Set (conjunto) no pueden estar vacíos. Las solicitudes con valores vacíos se rechazan con la excepción `ValidationException`.

Cada entrada `AttributeUpdates` consta de un nombre de atributo que se va a modificar, junto con lo siguiente:

- `Value`: el nuevo valor, si procede, de este atributo.
- `Action`: un valor que especifica cómo llevar a cabo la actualización. Esta acción solo es válida para un atributo existente cuyo tipo de datos sea Number o que sea un conjunto; no use `ADD` con otros tipos de datos.

  Si se encuentra en la tabla un elemento con la clave principal especificada, los valores siguientes llevan a cabo estas acciones:

  - `PUT`: añade el atributo especificado al elemento. Si el atributo ya existe, se sustituye por el nuevo valor.
  - `DELETE`: elimina el atributo y su valor, si no se ha especificado ningún valor para `DELETE`. El tipo de datos del valor especificado debe coincidir con el tipo de datos del valor existente.

    Si se especifica un conjunto de valores, entonces esos valores se restarán del conjunto anterior. Por ejemplo, si el valor de atributo era el conjunto `[a,b,c]` y la acción `DELETE` especifica `[a,c]`, entonces el valor final del atributo es `[b]`. Especificar un conjunto vacío es un error.
  - `ADD`: si el atributo aún no existe, añade el valor especificado al elemento. Si el atributo existe, el comportamiento de `ADD` depende del tipo de datos del atributo:
    - Si el atributo existente es un número y `Value` también es un número, entonces `Value` se suma matemáticamente al atributo existente. Si `Value` es un número negativo, entonces se resta del atributo existente.

      > **Note**
      >
      > Si utiliza `ADD` para sumar o restar de un valor numérico de un elemento que no existía antes de la actualización, DynamoDB utilizará 0 como valor inicial.

De igual forma, si utiliza `ADD` con un elemento existente para sumar o restar de un valor de un atributo que no existía antes de la actualización, DynamoDB utilizará `0` como valor inicial. Por ejemplo, supongamos que el elemento que desea actualizar no tiene un atributo denominado itemcount, pero que, a pesar de ello, usted decide aplicar `ADD` para sumar `3` a este atributo. DynamoDB creará el atributo itemcount, establecerá su valor inicial en `0` y, por último, le sumará `3`. El resultado será un nuevo atributo itemcount, cuyo valor será `3`.

- Si el tipo de datos existente es un conjunto y, además, `Value` también es un conjunto, `Value` se adjunta al conjunto existente. Por ejemplo, si el valor del atributo es el conjunto `[1,2]` y la acción `ADD` especifica `[3]`, entonces el valor final del atributo es `[1,2,3]`. Se produce un error si se especifica una acción `ADD` para un atributo de tipo Set y el tipo de atributo especificado no coincide con el tipo de conjunto existente.

  Ambos conjuntos deben tener el mismo tipo de datos primitivo. Por ejemplo, si el tipo de datos existente es un conjunto de cadenas, `Value` también debe ser un conjunto de cadenas.

Si no se encuentra en la tabla ningún elemento con la clave especificada, los valores siguientes llevan a cabo estas acciones:

- `PUT`: hace que DynamoDB cree un nuevo elemento con la clave principal especificada y, a continuación, agrega el atributo.
- `DELETE`: no sucede nada, porque no se pueden eliminar los atributos de un elemento que no existe. La operación se lleva a cabo, pero DynamoDB no crea ningún elemento nuevo.
- `ADD`: hace que DynamoDB cree un elemento con la clave principal y el número (o el conjunto de números) suministrados para el valor del atributo. Los únicos tipos de datos permitidos son Number y Number Set.

Si proporciona atributos que forman parte de una clave de índice, entonces los tipos de datos de tales atributos deberán coincidir con los del esquema de la definición de atributos de la tabla.

Use UpdateExpression en su lugar; ejemplo

Supongamos que desea modificar un elemento de la tabla Music. Podría usar una solicitud `UpdateItem` con un parámetro `AttributeUpdates` como en este ejemplo de la AWS CLI:

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
```

```
        "SongTitle": {"S":"Call Me Today"},
        "Artist": {"S":"No One You Know"}
    }' \
    --attribute-updates '{
        "Genre": {
            "Action": "PUT",
            "Value": {"S":"Rock"}
        }
    }'
```

Puede utilizar `UpdateExpression` en su lugar:

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "SongTitle": {"S":"Call Me Today"},
        "Artist": {"S":"No One You Know"}
    }' \
    --update-expression 'SET Genre = :g' \
    --expression-attribute-values '{
        ":g": {"S":"Rock"}
    }'
```

ConditionalOperator (heredado)

> **Note**
>
> Le recomendamos que utilice los nuevos parámetros de expresión en lugar de estos parámetros heredados siempre que sea posible. Para obtener más información, consulte Uso de expresiones en DynamoDB.

El parámetro condicional heredado `ConditionalOperator` es un operador lógico que se utiliza para aplicar a las condiciones de `Expected`, `QueryFilter` o mapa de `ScanFilter`:

- AND: si todas las condiciones se evalúan en true, entonces todo el mapa se evalúa en true.
- OR: si al menos una de las condiciones se evalúa en true, entonces todo el mapa se evalúa en true.

Si se omite `ConditionalOperator`, entonces `AND` es el valor predeterminado.

La operación se llevará a cabo correctamente solo si todo el mapa se evalúa en true.

> **Note**
>
> Este parámetro no es compatible con atributos de tipo List o Map.

Expected (heredado)

> **Note**
>
> Le recomendamos que utilice los nuevos parámetros de expresión en lugar de estos parámetros heredados siempre que sea posible. Para obtener más información, consulte Uso de expresiones en DynamoDB. Para obtener información específica sobre el nuevo parámetro que reemplaza a este, Use ConditionExpression en su lugar..

El parámetro condicional heredado `Expected` es un bloque condicional para una operación `UpdateItem`. `Expected` es un mapa de pares atributo-condición. Cada entrada del mapa consta de un nombre de atributo, un operador de comparación y uno o varios valores. DynamoDB utiliza el operador de comparación para comparar el atributo con el o los valores suministrados. Para cada entrada de `Expected`, el resultado de la evaluación es true o false.

Si especifica más de una entrada en el mapa `Expected`, de forma predeterminada todas las condiciones deben evaluarse en true (verdadero). Es decir, las condiciones se combinan con el operador AND. (Si lo desea, puede usar el parámetro `ConditionalOperator` para definir las condiciones en OR (o). En tal caso, deberá evaluarse en true (verdadero) al menos una de las condiciones, en lugar de todas ellas).

Si el mapa `Expected` se evalúa en true, entonces la operación condicional se realiza correctamente; de lo contrario, se produce un error.

`Expected` contiene lo siguiente:

- `AttributeValueList`: uno o más valores que se evaluarán respecto al atributo suministrado. El número de valores de la lista depende del valor de `ComparisonOperator` que se utilice.

  Para el tipo Number, las comparaciones de los valores son numéricas.

Las comparaciones de valores String (cadena) de tipo mayor que, igual que o menor que se basan en Unicode con la codificación UTF-8 binaria. Por ejemplo, a es mayor que A y a es mayor que B.

Al comparar valores de tipo Binary, DynamoDB trata cada byte de los datos binarios como sin signo.

- `ComparisonOperator`: comparador que permite evaluar los atributos contenidos en `AttributeValueList`. Al realizar la comparación, DynamoDB utiliza lecturas de consistencia alta.

  Están disponibles los siguientes operadores de comparación:

  `EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS | BEGINS_WITH | IN | BETWEEN`

  A continuación se indican las descripciones de cada operador de comparación.

  - `EQ`: igual. `EQ` se admite para todos los tipos de datos, incluyendo las listas y los mapas.

    `AttributeValueList` puede contener una sola entrada `AttributeValue` de tipo String (cadena), Number (número), Binary (binario), String Set (conjunto de cadenas), Number Set (conjunto de números) o Binary Set (conjunto de binarios). Si un elemento contiene una entrada `AttributeValue` de un tipo distinto del proporcionado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}`. `{"N":"6"}` tampoco es igual que `{"NS":["6", "2", "1"]}`.

  - `NE`: distinto. `NE` se admite para todos los tipos de datos, incluyendo las listas y los mapas.

    `AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena), Number (número), Binary (binario), String Set (conjunto de cadenas), Number Set (conjunto de números) o Binary Set (conjunto de binarios). Si un elemento contiene un `AttributeValue` de un tipo distinto del proporcionado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}`. `{"N":"6"}` tampoco es igual que `{"NS":["6", "2", "1"]}`.

  - `LE`: menor o igual que.

    `AttributeValueList` puede contener una sola entrada `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no de tipo Set [conjunto]). Si un elemento contiene una entrada `AttributeValue` de un tipo distinto del proporcionado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}`. `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.

- `LT` : Menor que.

  `AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no de tipo Set [conjunto]). Si un elemento contiene una entrada `AttributeValue` de un tipo distinto del proporcionado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}`. `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.

- `GE`: mayor o igual que.

  `AttributeValueList` puede contener una sola entrada `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no de tipo Set [conjunto]). Si un elemento contiene una entrada `AttributeValue` de un tipo distinto del proporcionado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}`. `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.

- `GT` : Mayor que.

  `AttributeValueList` puede contener una sola entrada `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no de tipo Set [conjunto]). Si un elemento contiene una entrada `AttributeValue` de un tipo distinto del proporcionado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}`. `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.

- `NOT_NULL`: el atributo existe. `NOT_NULL` se admite para todos los tipos de datos, incluyendo las listas y los mapas.

  > **Note**
  >
  > Este operador comprueba si existe un atributo, no su tipo de datos. Si el tipo de datos del atributo “a” es null y se evalúa mediante `NOT_NULL`, el resultado es un valor `true` booleano. Este resultado se debe a que el atributo "a" existe; su tipo de datos no es pertinente para el operador de comparación `NOT_NULL`.

- `NULL`: el atributo no existe. `NULL` se admite para todos los tipos de datos, incluyendo las listas y los mapas.

> **Note**
>
> Este operador comprueba la inexistencia de un atributo, no su tipo de datos. Si el tipo de datos del atributo “a” es null y se evalúa mediante `NULL`, el resultado es un valor `false` booleano. Esto se debe a que el atributo "a" existe; su tipo de datos no es pertinente para el operador de comparación `NULL`.

- `CONTAINS`: comprueba si hay una subsecuencia, o un valor en un conjunto.

  `AttributeValueList` puede contener una sola entrada `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no de tipo Set [conjunto]). Si el atributo de destino de la comparación es de tipo String (cadena), entonces el operador comprueba si hay una subcadena coincidente. Si el atributo de destino de la comparación es de tipo Binary (binario), entonces el operador busca una subsecuencia del destino que coincida con la entrada. Si el atributo de destino de la comparación es un conjunto ("SS", "NS" o "BS"), entonces el operador se evalúa en true si encuentra una coincidencia exacta con cualquier miembro del conjunto.

  CONTAINS es compatible con listas: al evaluar "a `CONTAINS` b", "a" puede ser una lista; sin embargo, "b" no puede ser un conjunto, un mapa ni una lista.

- `NOT_CONTAINS`: comprueba la ausencia de una subsecuencia, o la ausencia de un valor en un conjunto.

  `AttributeValueList` puede contener una sola entrada `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no de tipo Set [conjunto]). Si el atributo de destino de la comparación es de tipo String (cadena), entonces el operador comprueba la ausencia de una subcadena coincidente. Si el atributo de destino de la comparación es de tipo Binary (binario), entonces el operador busca la ausencia de una subsecuencia del destino que coincida con la entrada. Si el atributo de destino de la comparación es un conjunto ("SS", "NS" o "BS"), entonces el operador se evalúa en true si no (`does not`) encuentra una coincidencia exacta con cualquier miembro del conjunto.

  NOT_CONTAINS es compatible con listas: al evaluar "a `NOT CONTAINS` b", "a" puede ser una lista; sin embargo, "b" no puede ser un conjunto, un mapa ni una lista.

- `BEGINS_WITH`: comprueba si hay un prefijo.

`AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena) o Binary (binario) (no de tipo Number [número] ni Set [conjunto]). El atributo de destino de la comparación debe ser un valor de tipo String o Binary (no de tipo Number ni Set).

- `IN`: comprueba si hay entradas que coincidan en dos conjuntos.

  `AttributeValueList` puede contener una o varias entradas `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no de tipo Set [conjunto]). Estos atributos se comparan con un atributo existente de tipo Set (conjunto) de un elemento. Si cualquier entrada del conjunto de entrada está presente en el atributo del elemento, la expresión se evalúa en true.

- `BETWEEN`: mayor o igual que el primer valor y menor o igual que el segundo valor.

  `AttributeValueList` debe contener dos entradas `AttributeValue` del mismo tipo, que puede ser String (cadena), Number (número) o Binary (binario) (pero no de tipo Set [conjunto]). Un atributo de destino coincide si el valor de destino es mayor o igual que la primer entrada y menor o igual que la segunda entrada. Si un elemento contiene una entrada `AttributeValue` de un tipo distinto del proporcionado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no se compara con `{"N":"6"}`. `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.

Los parámetros siguientes se pueden utilizar en lugar de `AttributeValueList` y `ComparisonOperator`:

- `Value`: valor que DynamoDB compara con un atributo.
- `Exists`: valor de tipo Boolean que hace que DynamoDB evalúe el valor antes de intentar la operación condicional.
  - Si `Exists` es `true`, DynamoDB verificará si el valor del atributo ya existe en la tabla. Si lo encuentra, la condición se evalúa en true; de lo contrario, la condición se evalúa en false.
  - Si `Exists` es `false`, DynamoDB presupone que el valor del atributo `not` (no) existe en la tabla. Si el valor no existe, entonces el supuesto es válido y la condición se evalúa en true (verdadero). Si el valor se encuentra, a pesar del supuesto de que no existía, la condición se evalúa en false.

  Tenga en cuenta que el valor predeterminado de `Exists` es `true`.

Los parámetros `Value` y `Exists` son incompatibles con `AttributeValueList` y `ComparisonOperator`. Tenga en cuenta que si utiliza los dos conjuntos de parámetros a la vez, DynamoDB devolverá una excepción `ValidationException`.

> **Note**
>
> Este parámetro no es compatible con atributos de tipo List o Map.

Use ConditionExpression en su lugar; ejemplo

Supongamos que desea modificar un elemento de la tabla Music, pero solo si se cumple una condición determinada. Podría usar una solicitud `UpdateItem` con un parámetro `Expected` como en este ejemplo de la AWS CLI:

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
    }' \
    --attribute-updates '{
        "Price": {
            "Action": "PUT",
            "Value": {"N":"1.98"}
        }
    }' \
    --expected '{
        "Price": {
            "ComparisonOperator": "LE",
            "AttributeValueList": [ {"N":"2.00"} ]
        }
    }'
```

Puede utilizar `ConditionExpression` en su lugar:

```
aws dynamodb update-item \
    --table-name Music \
    --key '{
        "Artist": {"S":"No One You Know"},
        "SongTitle": {"S":"Call Me Today"}
```

```
    }' \
    --update-expression 'SET Price = :p1' \
    --condition-expression 'Price <= :p2' \
    --expression-attribute-values '{
        ":p1": {"N":"1.98"},
        ":p2": {"N":"2.00"}
    }'
```

KeyConditions (heredado)

> **Note**
>
> Le recomendamos que utilice los nuevos parámetros de expresión en lugar de estos parámetros heredados siempre que sea posible. Para obtener más información, consulte Uso de expresiones en DynamoDB. Para obtener información específica sobre el nuevo parámetro que reemplaza a este, Use KeyConditionExpression en su lugar..

El parámetro condicional heredado `KeyConditions` contiene los criterios de selección de una operación `Query`. Para ejecutar una consulta en una tabla, solo puede usar condiciones con los atributos de clave principal de la tabla. Debe proporcionar el nombre y valor de la clave de partición como una condición `EQ`. Si lo desea, puede proporcionar una segunda condición referida a la clave de ordenación.

> **Note**
>
> Si no proporciona una condición de clave de ordenación, se recuperarán todos los elementos que coincidan con la clave de partición. Si hay una expresión `FilterExpression` o `QueryFilter` presente, se aplicará después de recuperar los elementos.

Para ejecutar una consulta en un índice, solo puede usar condiciones con los atributos de clave del índice. Debe proporcionar el nombre y valor de la clave de partición del índice como una condición `EQ`. Si lo desea, puede proporcionar una segunda condición referida a la clave de ordenación del índice.

Cada entrada `KeyConditions` consta de un nombre de atributo que se va a comparar, junto con lo siguiente:

- `AttributeValueList`: uno o más valores que se evaluarán respecto al atributo suministrado. El número de valores de la lista depende del valor de `ComparisonOperator` que se utilice.

  Para el tipo Number, las comparaciones de los valores son numéricas.

  Las comparaciones de valores String (cadena) de tipo mayor que, igual que o menor que se basan en Unicode con la codificación UTF-8 binaria. Por ejemplo, `a` es mayor que `A` y `a` es mayor que `B`.

  Al comparar valores de tipo Binary, DynamoDB trata cada byte de los datos binarios como sin signo.

- `ComparisonOperator`: comparador que permite evaluar los atributos. Por ejemplo: igual que, mayor que y menor que.

  Para `KeyConditions`, solo se admiten los siguientes operadores de comparación:

  `EQ | LE | LT | GE | GT | BEGINS_WITH | BETWEEN`

  A continuación se indican las descripciones de estos operadores de comparación.

  - `EQ`: igual que.

    `AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no de tipo Set [conjunto]). Si un elemento contiene una entrada `AttributeValue` de un tipo distinto del especificado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}`. `{"N":"6"}` tampoco es igual que `{"NS":["6", "2", "1"]}`.

  - `LE`: menor o igual que.

    `AttributeValueList` puede contener una sola entrada `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no de tipo Set [conjunto]). Si un elemento contiene una entrada `AttributeValue` de un tipo distinto del proporcionado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}`. `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.

  - `LT` : Menor que.

    `AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no de tipo Set [conjunto]). Si un elemento contiene una entrada `AttributeValue` de un tipo distinto del proporcionado en la solicitud, el valor no

coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}`. `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.

- `GE`: mayor o igual que.

  `AttributeValueList` puede contener una sola entrada `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no de tipo Set [conjunto]). Si un elemento contiene una entrada `AttributeValue` de un tipo distinto del proporcionado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}`. `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.

- `GT` : Mayor que.

  `AttributeValueList` puede contener una sola entrada `AttributeValue` de tipo String (cadena), Number (número) o Binary (binario) (no de tipo Set [conjunto]). Si un elemento contiene una entrada `AttributeValue` de un tipo distinto del proporcionado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no es igual que `{"N":"6"}`. `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.

- `BEGINS_WITH`: comprueba si hay un prefijo.

  `AttributeValueList` puede contener un solo `AttributeValue` de tipo String (cadena) o Binary (binario) (no de tipo Number [número] ni Set [conjunto]). El atributo de destino de la comparación debe ser un valor de tipo String o Binary (no de tipo Number ni Set).

- `BETWEEN`: mayor o igual que el primer valor y menor o igual que el segundo valor.

  `AttributeValueList` debe contener dos entradas `AttributeValue` del mismo tipo, que puede ser String (cadena), Number (número) o Binary (binario) (pero no de tipo Set [conjunto]). Un atributo de destino coincide si el valor de destino es mayor o igual que la primer entrada y menor o igual que la segunda entrada. Si un elemento contiene una entrada `AttributeValue` de un tipo distinto del proporcionado en la solicitud, el valor no coincide. Por ejemplo, `{"S":"6"}` no se compara con `{"N":"6"}`. `{"N":"6"}` tampoco se compara con `{"NS":["6", "2", "1"]}`.

Use KeyConditionExpression en su lugar; ejemplo

Supongamos que desea recuperar varios elementos con la misma clave de partición de la tabla Music. Podría usar una solicitud `Query` con un parámetro `KeyConditions` como en este ejemplo de la AWS CLI:

```
aws dynamodb query \
    --table-name Music \
    --key-conditions '{
        "Artist":{
            "ComparisonOperator":"EQ",
            "AttributeValueList": [ {"S": "No One You Know"} ]
        },
        "SongTitle":{
            "ComparisonOperator":"BETWEEN",
            "AttributeValueList": [ {"S": "A"}, {"S": "M"} ]
        }
    }'
```

Puede utilizar `KeyConditionExpression` en su lugar:

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression 'Artist = :a AND SongTitle BETWEEN :t1 AND :t2' \
    --expression-attribute-values '{
        ":a": {"S": "No One You Know"},
        ":t1": {"S": "A"},
        ":t2": {"S": "M"}
    }'
```

QueryFilter (heredado)

> **Note**
>
> Le recomendamos que utilice los nuevos parámetros de expresión en lugar de estos parámetros heredados siempre que sea posible. Para obtener más información, consulte Uso de expresiones en DynamoDB. Para obtener información específica sobre el nuevo parámetro que reemplaza a este, Use FilterExpression en su lugar..

En una operación `Query`, el parámetro condicional heredado `QueryFilter` es una condición que evalúa los resultados de la consulta después de que se hayan leído los elementos y devuelve únicamente los valores deseados.

Este parámetro no es compatible con atributos de tipo List o Map.

> **Note**
>
> `QueryFilter` se aplica después de haber leído los elementos; el proceso de filtrado no consume ninguna unidad de capacidad de lectura adicional.

Si proporciona más de una condición en el mapa `QueryFilter`, de forma predeterminada todas las condiciones deben evaluarse en true (verdadero). Es decir, las condiciones se combinan con el operador `AND`. (Si lo desea, puede usar el parámetro ConditionalOperator (heredado) para definir las condiciones en OR (o). En tal caso, deberá evaluarse en true (verdadero) al menos una de las condiciones, en lugar de todas ellas).

Tenga en cuenta que `QueryFilter` no admite atributos de clave. No puede definir una condición de filtro en una clave de partición ni de ordenación.

Cada entrada `QueryFilter` consta de un nombre de atributo que se va a comparar, junto con lo siguiente:

- `AttributeValueList`: uno o más valores que se evaluarán respecto al atributo suministrado. El número de valores de la lista depende del operador especificado en `ComparisonOperator`.

  Para el tipo Number, las comparaciones de los valores son numéricas.

  Las comparaciones de valores String (cadena) de tipo mayor que, igual que o menor que se basan en la codificación UTF-8 binaria. Por ejemplo, `a` es mayor que `A` y `a` es mayor que `B`.

  Al comparar valores de tipo Binary, DynamoDB trata cada byte de los datos binarios como sin signo.

  Para obtener más información sobre cómo especificar los tipos de datos en JSON, consulte API de bajo nivel de DynamoDB.
- `ComparisonOperator`: comparador que permite evaluar los atributos. Por ejemplo: igual que, mayor que y menor que.

  Están disponibles los siguientes operadores de comparación:

  ```
  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS |
  BEGINS_WITH | IN | BETWEEN
  ```

Use FilterExpression en su lugar; ejemplo

Supongamos que quiere consultar la tabla Music y aplicar una condición a la coincidencia de elementos. Podría usar una solicitud `Query` con un parámetro `QueryFilter` como en este ejemplo de la AWS CLI:

```
aws dynamodb query \
    --table-name Music \
    --key-conditions '{
        "Artist": {
            "ComparisonOperator": "EQ",
            "AttributeValueList": [ {"S": "No One You Know"} ]
        }
    }' \
    --query-filter '{
        "Price": {
            "ComparisonOperator": "GT",
            "AttributeValueList": [ {"N": "1.00"} ]
        }
    }'
```

Puede utilizar `FilterExpression` en su lugar:

```
aws dynamodb query \
    --table-name Music \
    --key-condition-expression 'Artist = :a' \
    --filter-expression 'Price > :p' \
    --expression-attribute-values '{
        ":p": {"N":"1.00"},
        ":a": {"S":"No One You Know"}
    }'
```

ScanFilter (heredado)

> **Note**
>
> Le recomendamos que utilice los nuevos parámetros de expresión en lugar de estos parámetros heredados siempre que sea posible. Para obtener más información, consulte Uso de expresiones en DynamoDB. Para obtener información específica sobre el nuevo parámetro que reemplaza a este, Use FilterExpression en su lugar..

En una operación `Scan`, el parámetro condicional heredado `ScanFilter` es una condición que evalúa los resultados del examen y devuelve únicamente los valores deseados.

> **Note**
>
> Este parámetro no es compatible con atributos de tipo List o Map.

Si especifica más de una condición en el mapa `ScanFilter`, de forma predeterminada todas las condiciones deben evaluarse en true (verdadero). Es decir, se utiliza AND como operador para evaluar las condiciones. (Si lo desea, puede usar el parámetro ConditionalOperator (heredado) para definir las condiciones en OR (o). En tal caso, deberá evaluarse en true (verdadero) al menos una de las condiciones, en lugar de todas ellas).

Cada entrada `ScanFilter` consta de un nombre de atributo que se va a comparar, junto con lo siguiente:

- `AttributeValueList`: uno o más valores que se evaluarán respecto al atributo suministrado. El número de valores de la lista depende del operador especificado en `ComparisonOperator`.

  Para el tipo Number, las comparaciones de los valores son numéricas.

  Las comparaciones de valores String (cadena) de tipo mayor que, igual que o menor que se basan en la codificación UTF-8 binaria. Por ejemplo, `a` es mayor que `A` y `a` es mayor que `B`.

  Al comparar valores de tipo Binary, DynamoDB trata cada byte de los datos binarios como sin signo.

  Para obtener más información sobre cómo especificar los tipos de datos en JSON, consulte API de bajo nivel de DynamoDB.
- `ComparisonOperator`: comparador que permite evaluar los atributos. Por ejemplo: igual que, mayor que y menor que.

  Están disponibles los siguientes operadores de comparación:

  ```
  EQ | NE | LE | LT | GE | GT | NOT_NULL | NULL | CONTAINS | NOT_CONTAINS |
  BEGINS_WITH | IN | BETWEEN
  ```

Use FilterExpression en su lugar; ejemplo

Supongamos que quiere examinar la tabla Music y aplicar una condición a la coincidencia de elementos. Podría usar una solicitud `Scan` con un parámetro `ScanFilter` como en este ejemplo de la AWS CLI:

```
aws dynamodb scan \
    --table-name Music \
    --scan-filter '{
        "Genre":{
            "AttributeValueList":[ {"S":"Rock"} ],
            "ComparisonOperator": "EQ"
        }
    }'
```

Puede utilizar `FilterExpression` en su lugar:

```
aws dynamodb scan \
    --table-name Music \
    --filter-expression 'Genre = :g' \
    --expression-attribute-values '{
        ":g": {"S":"Rock"}
    }'
```

Escritura de condiciones con parámetros heredados

> **Note**
>
> Le recomendamos que utilice los nuevos parámetros de expresión en lugar de estos parámetros heredados siempre que sea posible. Para obtener más información, consulte Uso de expresiones en DynamoDB.

En la siguiente sección se describe cómo escribir condiciones para usarlas con parámetros heredados, tales como `Expected`, `QueryFilter` y `ScanFilter`.

> **Note**
>
> En las nuevas aplicaciones deben utilizarse parámetros de expresión en su lugar. Para obtener más información, consulte Uso de expresiones en DynamoDB.

Condiciones simples

Con los valores de los atributos, puede escribir condiciones para realizar comparaciones con los atributos de la tabla. Una condición siempre se evalúa en true o false y consta de:

- `ComparisonOperator`: mayor que, menor que, igual que, etc.
- `AttributeValueList` (opcional): valor o valores de atributo respecto a los que se va a establecer la comparación. Según cuál esa el operador `ComparisonOperator` utilizado, `AttributeValueList` puede contener uno, dos o más valores; o puede omitirse.

En las siguientes secciones se describen los distintos operadores de comparación, además de ejemplos de cómo utilizarlos en las condiciones.

Operadores de comparación sin valores de atributos

- `NOT_NULL`: es true si un atributo existe.
- `NULL`: es true si un atributo no existe.

Utilice estos operadores para verificar si un atributo existe o no. Dado que no hay ningún valor que comparar, no especifique `AttributeValueList`.

Ejemplo

La siguiente expresión se evalúa en true si el atributo Dimensions existe.

```
...
    "Dimensions": {
         ComparisonOperator: "NOT_NULL"
    }
...
```

Operadores de comparación con un valor de atributo

- `EQ`: es true si un atributo es igual que un valor.

  `AttributeValueList` puede contener un solo valor de tipo String (cadena), Number (número), Binary (binario), String Set (conjunto de cadenas), Number Set (conjunto de números) o Binary Set (conjunto de binarios). Si un elemento contiene un valor de un tipo distinto del especificado en la solicitud, el valor no coincide. Por ejemplo, la cadena `"3"` no es igual que el número `3`. Tampoco el número `3` es igual que el conjunto de números `[3, 2, 1]`.

- `NE`: es true si un atributo no es igual que un valor.

  `AttributeValueList` puede contener un solo valor de tipo String (cadena), Number (número), Binary (binario), String Set (conjunto de cadenas), Number Set (conjunto de números) o Binary Set (conjunto de binarios). Si un elemento contiene un valor de un tipo distinto del especificado en la solicitud, el valor no coincide.

- `LE`: es true si un atributo es menor o igual que un valor.

  `AttributeValueList` puede contener un solo valor de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si un elemento contiene un `AttributeValue` de un tipo distinto del especificado en la solicitud, el valor no coincide.

- `LT`: es true si un atributo es menor que un valor.

  `AttributeValueList` puede contener un solo valor de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si un elemento contiene un valor de un tipo distinto del especificado en la solicitud, el valor no coincide.

- `GE`: es true si un atributo es mayor o igual que un valor.

  `AttributeValueList` puede contener un solo valor de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si un elemento contiene un valor de un tipo distinto del especificado en la solicitud, el valor no coincide.

- `GT`: es true si un atributo es mayor que un valor.

  `AttributeValueList` puede contener un solo valor de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si un elemento contiene un valor de un tipo distinto del especificado en la solicitud, el valor no coincide.

- `CONTAINS`: es true si un valor está presente en un conjunto o si un valor contiene otro valor.

`AttributeValueList` puede contener un solo valor de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si el atributo de destino de la comparación es de tipo String (cadena), entonces el operador comprueba si hay una subcadena coincidente. Si el atributo de destino de la comparación es de tipo Binary (binario), entonces el operador busca una subsecuencia del destino que coincida con la entrada. Si el atributo de destino de la comparación es un conjunto, entonces el operador se evalúa en true si encuentra una coincidencia exacta con cualquier miembro del conjunto.

- `NOT_CONTAINS`: es true si un valor no está presente en un conjunto o si un valor no contiene otro valor.

  `AttributeValueList` puede contener un solo valor de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Si el atributo de destino de la comparación es de tipo String (cadena), entonces el operador comprueba la ausencia de una subcadena coincidente. Si el atributo de destino de la comparación es de tipo Binary (binario), entonces el operador busca la ausencia de una subsecuencia del destino que coincida con la entrada. Si el atributo de destino de la comparación es un conjunto, entonces el operador se evalúa en true si no encuentra una coincidencia exacta con cualquier miembro del conjunto.
- `BEGINS_WITH`: es true (verdadero) si los primeros caracteres de un atributo coinciden con el valor proporcionado. No use este operador para comparar números.

  `AttributeValueList` puede contener un solo valor de tipo String (cadena) o Binary (binario) (no de tipo Number [número] o Set [conjunto]). El atributo de destino de la comparación debe ser un valor de tipo String o Binary (no de tipo Number ni un conjunto).

Use estos operadores para comparar un atributo con un valor. Debe especificar una lista `AttributeValueList` compuesta por un solo valor. Para la mayoría de los operadores, este debe ser un valor escalar; sin embargo, los operadores `EQ` y `NE` también admiten conjuntos.

Ejemplos

Las siguientes expresiones se evalúan en true si:

- El precio de un producto es mayor que 100.

```
...
    "Price": {
        ComparisonOperator: "GT",
        AttributeValueList: [ {"N":"100"} ]
```

```
    }
...
```

- La categoría de un producto comienza con "Bo".

```
...
    "ProductCategory": {
        ComparisonOperator: "BEGINS_WITH",
        AttributeValueList: [ {"S":"Bo"} ]
    }
...
```

- Un producto está disponible en rojo, verde o negro:

```
...
    "Color": {
        ComparisonOperator: "EQ",
        AttributeValueList: [
            [ {"S":"Black"}, {"S":"Red"}, {"S":"Green"} ]
        ]
    }
...
```

> **Note**
>
> Al comparar datos de tipo Set (conjunto), el orden de las entradas carece de relevancia. DynamoDB devolverá solo los elementos con el mismo conjunto de valores, independientemente del orden en que los especifique en la solicitud.

Operadores de comparación con dos valores de atributos

- `BETWEEN`: es true si un valor está comprendido entre un límite inferior y un límite superior, ambos incluidos.

  `AttributeValueList` debe contener dos entradas del mismo tipo, que puede ser String (cadena), Number (número) o Binary (binario) (pero no un conjunto). Un atributo de destino coincide si el valor de destino es mayor o igual que la primer entrada y menor o igual que la segunda entrada. Si un elemento contiene un valor de un tipo distinto del especificado en la solicitud, el valor no coincide.

Use este operador para determinar si un valor de atributo está dentro de un intervalo. `AttributeValueList` debe contener dos entradas escalares del mismo tipo: String, Number o Binary.

Ejemplo

La siguiente expresión se evalúa en true si el precio de un producto está comprendido entre 100 y 200.

```
...
    "Price": {
        ComparisonOperator: "BETWEEN",
        AttributeValueList: [ {"N":"100"}, {"N":"200"} ]
    }
...
```

Operadores de comparación con n valores de atributos

- `IN`: es true (verdadero) si un valor es igual que cualquiera de los valores de una lista. Solo se admiten valores escalares en la lista, no conjuntos. El atributo de destino debe ser del mismo tipo y valor exactos para que se produzca la coincidencia.

  `AttributeValueList` puede contener una o varias entradas de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). Estos atributos se comparan con un atributo existente que no sea de tipo Set (conjunto) de un elemento. Si cualquier entrada del conjunto de entrada está presente en el atributo del elemento, la expresión se evalúa en true.

  `AttributeValueList` puede contener uno o varios valores de tipo String (cadena), Number (número) o Binary (binario) (no un conjunto). El atributo de destino de la comparación debe ser del mismo tipo y valor exacto para que se produzca la coincidencia. Un valor de tipo String nunca coincide con un valor de tipo String Set.

Use este operador para determinar si el valor suministrado está contenido en una lista. Puede especificar cualquier número de valores escalares en `AttributeValueList`, pero todos ellos deben ser del mismo tipo de datos.

Ejemplo

La siguiente expresión se evalúa en true si el valor de Id es 201, 203 o 205.

```
...
    "Id": {
        ComparisonOperator: "IN",
        AttributeValueList: [ {"N":"201"}, {"N":"203"}, {"N":"205"} ]
    }
...
```

Uso de varias condiciones

DynamoDB permite combinar varias condiciones para crear expresiones complejas. Para ello, hay que proporcionar al menos dos expresiones, con un ConditionalOperator (heredado) opcional.

De forma predeterminada, cuando se especifica más de una condición, todas ellas deben evaluarse en true (verdadero) para que la expresión completa se evalúe en true (verdadero). Es decir, se lleva a cabo una operación AND implícita.

Ejemplo

La siguiente expresión se evalúa en true si un producto es un libro que tiene al menos 600 páginas. Deben evaluarse en true las dos condiciones, puesto que implícitamente se les aplica el operador AND.

```
...
    "ProductCategory": {
        ComparisonOperator: "EQ",
        AttributeValueList: [ {"S":"Book"} ]
    },
    "PageCount": {
        ComparisonOperator: "GE",
        AttributeValueList: [ {"N":600} ]
    }
...
```

Puede utilizar ConditionalOperator (heredado) para aclarar que se llevará a cabo una operación AND. En el siguiente ejemplo, el comportamiento es el mismo que en el caso anterior.

```
...
    "ConditionalOperator" : "AND",
    "ProductCategory": {
        "ComparisonOperator": "EQ",
```

```
        "AttributeValueList": [ {"N":"Book"} ]
    },
    "PageCount": {
        "ComparisonOperator": "GE",
        "AttributeValueList": [ {"N":600"} ]
    }
...
```

También puede establecer `ConditionalOperator` en OR, en cuyo caso al menos una de las condiciones deberá evaluarse en true.

Ejemplo

La siguiente expresión se evalúa en true si un producto es una bicicleta de montaña, es de una marca concreta o su precio es mayor que 100.

```
...
    ConditionalOperator : "OR",
    "BicycleType": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Mountain" ]
    },
    "Brand": {
        "ComparisonOperator": "EQ",
        "AttributeValueList": [ {"S":"Brand-Company A" ]
    },
    "Price": {
        "ComparisonOperator": "GT",
        "AttributeValueList": [ {"N":"100"} ]
    }
...
```

> **Note**
>
> En una expresión compleja, las condiciones se procesan por orden, desde la primera hasta la última.
>
> No se puede utilizar AND y OR en la misma expresión.

Otros operadores condicionales

En versiones anteriores de DynamoDB, el parámetro `Expected` se comportaba de forma distinta para las escrituras condicionales. Cada elemento del mapa `Expected` representaba un nombre de atributo que DynamoDB debía verificar, además de lo siguiente:

- `Value`: valor que se va a comparar con el atributo.
- `Exists`: determina si el valor existe antes de intentar la operación.

DynamoDB continúa admitiendo las opciones `Value` y `Exists`. Sin embargo, solo permiten comprobar una condición de desigualdad o si un atributo existe. Recomendamos usar `ComparisonOperator` y `AttributeValueList` en su lugar, porque estas opciones permiten construir un abanico de condiciones mucho más amplio.

Example

`DeleteItem` permite verificar si un libro está descatalogado y eliminarlo únicamente si se cumple esta condición. A continuación se muestra un ejemplo de la AWS CLI en el que se usa una condición heredada:

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
    --expected '{
        "InPublication": {
            "Exists": true,
            "Value": {"BOOL":false}
        }
    }'
```

En el siguiente ejemplo se realiza la misma operación, pero sin la condición heredada:

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{
        "Id": {"N":"600"}
    }' \
```

```
    --expected '{
        "InPublication": {
            "ComparisonOperator": "EQ",
            "AttributeValueList": [ {"BOOL":false} ]
        }
    }'
```

Example

Una operación `PutItem` permite proteger los datos para evitar que se sobrescriba un elemento existente que tenga los mismos atributos de clave principal. A continuación se muestra un ejemplo de la en el que se usa una condición heredada:

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "Exists": false }
    }'
```

En el siguiente ejemplo se realiza la misma operación, pero sin la condición heredada:

```
aws dynamodb put-item \
    --table-name ProductCatalog \
    --item '{
      "Id": {"N":"500"},
        "Title": {"S":"Book 500 Title"}
    }' \
    --expected '{
        "Id": { "ComparisonOperator": "NULL" }
    }'
```

> **Note**
>
> Para las condiciones del mapa `Expected`, no use las opciones `Value` y `Exists` heredadas con `ComparisonOperator` y `AttributeValueList`. Si lo hace, se producirá un error en la escritura condicional.